TAMPERE UNIVERSITY OF TECHNOLOGY

Emma Luukka

Version Control in Multivendor Projects

Master's Thesis

Examiner: professor Tommi Mikkonen
Examiner and topic accepted
in the faculty council meeting of
Computing and Electrical Engineering
on April 3rd 2013

ABSTRACT

TAMPERE UNIVERSITY OF TECHNOLOGY
Master's Degree Programme in Information Technology
LUUKKA, EMMA: VERSION CONTROL IN MULTIVENDOR PROJECTS
Master of Science Thesis, 68 pages
December 2013
Major: Software Engineering
Examiner: professor Tommi Mikkonen, MSc Mika Torhola
Keywords: version control, multivendor project, project management

Today's businesses focus on their core competences and buy all other business functions from other companies. Multisourcing happens when a project is divided between more than one company. Some companies are created to purely answer to these outsourcing needs. Atostek Oy is one such company participating in many multisourcing projects.

While outsourcing might seem like an easy way to handle non-essential business functions, it does require planning, organizing and communication from the company doing the outsourcing as well as the company the function is outsourced to. This adds challenges to the project managers on all ends.

Version control is essential in all software projects. It is essential that each version is tracked and accessible. Nowadays software projects are often so large that it is important for multiple developers to collaborate on them and therefore have access to all versions of the software.

Often version control has been handled using a centralized version control system. In these cases each company participating in a multivendor project sets up their own version control system and work on only their part of the software. The interfaces between parts and all communication about the project, such as documents and contracts, are shared between companies via another method. Nowadays distributed version control systems are becoming increasingly popular and offer multiple alternative ways of setting up the version control mechanism within a company as well as between companies.

In this thesis, multivendor projects that Atostek Oy participates in are investigated. First a study is conducted to see how the multivendor projects are managed. Then three projects are focused on especially, because the projects have decided to investigate which version control mechanism would be the best to suit the project's needs. Based on these a checklist for managing the different aspects of project management and a comparison of version control system archtypes are created.

TIIVISTELMÄ

TAMPEREEN TEKNILLINEN YLIOPISTO
Tietotekniikan koulutusohjelma
LUUKKA, EMMA: VERSIONHALLINTA MONITOIMITTAJAPROJEKTEISSA
Diplomityö, 68 sivua
Joulukuu 2013
Pääaine: Ohjelmistotuotanto
Tarkastaja: professori Tommi Mikkonen, diplomi-insinööri Mika Torhola
Avainsanat: versionhallinta, monitoimittajaprojekti, projektinhallinta

Nykyään yhtiöt keskittyvät yhä enenevissä määrin ydinosaamiseensa ja ostavat muut toiminnot ulkopuolisilta yrityksiltä. Kun yksi projekti jaetaan useamman tahon tehtäväksi, puhutaan monitoimittajaprojekteista. Useiden yhtiöiden toiminta perustuu ulkoistettujen projektien toteuttamiseen. Atostek Oy on yksi näistä yhtiöistä. Se osallistuu moniin monitoimittajaprojekteihin.

Toimintojen ulkoistaminen saattaa vaikuttaa helpolta tavalta hoitaa yhtiön ydinosaamisen ulkopuolelle jäävät tukitoiminnot. Se vaatii kuitenkin paljon suunnittelua, organisointia ja kommunikointia sekä toiminnon ulkoistajalta että toiminnon toteuttajalta. Tästä aiheutuu lisähaasteita molempien osapuolien projektinhallinnalle.

Versionhallinta on oleellinen osa ohjelmistoprojekteja. Jokaisen version tulee olla tarvittaessa saatavilla. Nykyään ohjelmistoprojektit ovat niin suuria, että niiden toteuttamiseen vaaditaan useamman toteuttajan yhteistyötä. Jokaisen toteuttajan tulee päästä käsiksi kaikkiin ohjelmiston ja sen osioiden versioihin.

Versionhallinta hoidetaan yleensä käyttämällä keskitettyä versionhallintajärjestelmää. Tällöin jokainen projektiin osallistuva toimittaja pystyttää oman versionhallintajärjestelmänsä, joihin muilla ei ole pääsyä. Ohjelmiston eri toimittajien välillä olevat rajapinnat määritellään yhdessä. Projektiin liittyvä dokumentaatio ja muu kommunikointi hoidetaan erillään versionhallinnasta. Suosiotaan kasvattavat hajautetut versionhallintajärjestelmät tarjoavat vaihtoehtoja sille, miten versionhallinta hoidetaan yhtiön sisällä ja yhtiöiden välillä.

Tässä diplomityössä tutustutaan monitoimittajaprojekteihin, joissa Atostek Oy on osallisena. Alkuun toteutetaan tutkimus, jossa selvitetään, miten monitoimittajaprojektien projektinhallinta on hoidettu. Tämän perustella laaditaan muistilista projektinhallintaa varten. Sen jälkeen perehdytään tarkemmin kolmeen projektiin, joissa on pohdittu käytössä olevaa versionhallintamekanismia ja päädytty vaihtamaan se johonkin toiseen. Projekteissa esiin nousseiden näkökohtien valossa verrataan erilaisia versionhallintajärjestelmiä toisiinsa.

FOREWORD

It would not have been possible for me to write this thesis without the much appreciated help and support of the following people.

First and foremost I would like to thank my supervisors professor Tommi Mikkonen and MSc Mika Torhola for their guidance. Without them this thesis would not have seen the light of day.

I would also like to thank Atostek Oy for providing me with the subject and the time for writing this thesis. Everybody at Atostek Oy has shown interest in my thesis and its progression, which has helped me to keep at it. Thank you all.

Last, but definitely not least, I would like to thank my family and friends from the bottom of my heart for their love and support throughout my studies and the process of writing this thesis. You are all invaluable to me. I am especially grateful for the patience and understanding showed by Jukka Varjo, Jari Voutilainen and Kat Merican, who have followed the progression of this thesis very closely.

Tampere, November 11th, 2013

Emma Luukka

INDEX

1. INTRODUCTION

Multivendor projects are result when parts of business functions are outsourced to more than a single company. In them multiple companies work on different parts of the project. While multivendor projects are often seen as a way to save money for the customer company, it is not easy, and if not done right it can end up costing the company more than providing the same services in-house. This thesis focuses on the research and development of software.

There are multiple companies, such as Atostek Oy, that the research and development functions are outsourced to. These are the vendors in multivendor projects. Some companies develop devices, others develop software. Some do both.

Having multiple vendors work on the same project sets extra demands on project management. Communication is especially important. It needs to happen openly between all parties. The customer and all vendors need to work throughout the project if it is to be completed successfully.

When only one company handles the entire outsourced project, they can set up version control however they choose. However when multiple companies work on the same project this is not as easy. If each company has different version control systems, the interface between the parts developed by each company need to be very clearly defined. The version control system is rarely shared between different companies though the rise of distributed version control systems could change that.

In this thesis a study is conducted to find out how the current multivendor projects are managed in Atostek Oy. A checklist for project management is created based on the study. Then three of these projects are focused on as case studies. In each case a project in which the use of distributed version control system between all vendors is investigated. Based on these case studies and literature different version control systems are evaluated.

Chapter 2 introduces the different aspects of project management in the form of a literacy review. Chapter 3 explains the basic structure and function of version control and introduces different types of version control systems. In Chapter 4 the results of the study are analyzed and a checklist for project management is proposed. The case studies are discussed in Chapter 5. An evaluation of the different types of version control systems based on the case studies and literature is conducted in Chapter 6. Chapter 7 draws some final conclusions.

2. DIFFERENT ASPECTS OF PROJECT MANAGEMENT

Project management is often discussed as a whole, and often companies have people with Project Manager as their title. However project management is a very large concept with many different aspects. Despite the fact they are often treated as a group, it is benefitial to identify each aspect of project management. This way it is easier to ensure that each aspect is covered.

2.1 Background

One way of defining project management is provided by Effective Project Management: Traditional, Adaptive, Extreme [41]. According to it, traditional project management is composed of defining, planning, executing, controlling, and closing of the project.

After the definition phase of a project, the problem the project aims to solve, the goal of the project, the objectives to be met in order to accomplish the goal, the meters used to determine the success of the project, as well as the assumptions, risks and obstacles affecting the project should be clear.

Although there will almost always be changes to the project during its entire life cycle, it is important to plan the project. It reduces uncertainty by forcing the planners to consider different outcomes and thus be prepared for different scenarios. It increases understanding of the goals and tasks of the project because the planners need to focus on them in more detail. It also improves efficiency because by the end of the planning process there is an idea of the resources needed for each task and therefore it is easier to schedule the use of and to acquire the different resources needed.

After the planning is done it is time to execute the plan. This means identifying the resources (developers, equipment, money, and time to name a few) needed for the project. If some of the resources are in heavy use elsewhere, such as other projects, it is important to know when each resource is needed so it is easier to justify using the resource. After identifying and acquiring resources the tasks must be scheduled and assigned to developers. After this all that remains is to put the plan into action.

Controlling the project consists mainly of keeping track of the schedules and how well they are kept, as well as any other possible changes that might affect the project.

Change management can be seen as a separate aspect of project management or as part of continuous and active management of requirements and risks. In this thesis the latter approach is used.

At the end of the project a signal is given to mark the termination of the project. Usually this happens when the final product is given to the customer or when the customer says they are happy with the product they have received and the product has succesfully passed the customer's tests.

Project management is often divided into different aspects with different responsibilities. Most common ones are product management, requirements management, task management, configuration management, version control, test management, defect management and risk management. These cover the five project management aspects. Defining phase is part product management part requirements management and part risk management. Planning is covered by risk management, task management and configuration management. Configuration, task, test and defect management cover the execution phase. The controlling phase involves the constant following and updating of documents in all areas of project management. In the final stage the way each aspect of project management was handled are examined to see if anything can be learned.

The different aspects of project management are not completely isolated. In fact many of them are closely related. The relationships are depicted in Figure 2.1.

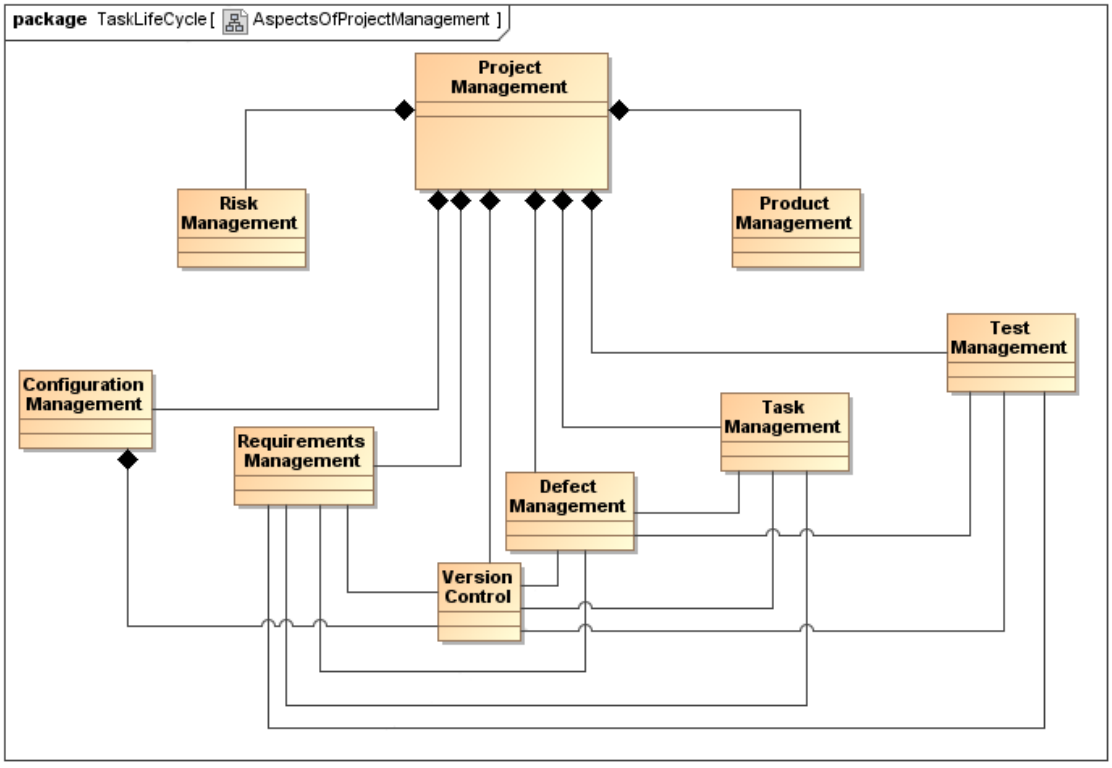

Figure 2.1: The relationships between different aspects of project management.

2.2 Product Management

As a marketing term a product is anything that can be pushed to a market that could satisfy a want or need [32]. It can be a physical thing such as computer mouse or an elevator or it can be immaterial such as software or human work.

Product management is used in every day conversations to mean different things. To some it is a synonym to configuration management whereas some consider it to be an aspect of marketing [27, 25]. There are also some who view it as a high level management that is a bit of everything: marketing, sales, implementation, development, accounting, legal and business development [42]. The main difference between these seemingly conflicting views is the person responsible for product management. If product management is considered to be synonymous to configuration management, the project manager responsible for the development and implementation of the product is also the product manager. If it is considered to be a part of marketing a marketing manager is also the product manager. In the last case there is a completely independent product manager who communicates and collaborates with the managers of different sections such as development and marketing. In all cases the goal is to make sure that the end product is what the customers want and will buy.

Hence product management is difficult to place in Figure 2.1. If it is considered to be a part of marketing, it would not even be in Figure 2.1 at all. On the other hand, it could be a part of configuration management. Because of the controversy it is its own thing, relating to project management but not connected to other aspects of it.

In multivendor projects the marketing aspect of product management is in a smaller role than in conventional projects since the customer has often already agreed to buy the end product. In these projects product management is closer to configuration and requirements management since it is important to figure out what it is exactly that the customer wants and how to get there. This includes knowing what the other vendors are doing and especially how the interfaces of their components work.

2.3 Requirements Management

A requirement is "a condition or capability needed by a stakeholder to solve a problem or achieve an objective" or "a condition or capability that must be met of possessed by a solution or solution component to satisfy a contract, standard, specification or other formally imposed documents" [20]. It is important to document requirements. So much so that IEEE has created a standard for it [43].

In A guide to the Business Analysis Body of Knowledge (BABOK guide), version

2.0 [20] requirements are divided into five different categories: business requirements, user requirements, functional requirements, non-functional requirements and implementation requirements. Business requirements state the goals, objectives or needs of an organization on a high level. The goal is to increase revenue, improve service or address other opportunities the organization wants realized, or avoid costs, meet regulatory requirements or state other problems the organization wants solved. User requirements or stakeholder requirements describe how a user or group of users want to interact with or use the end product and what needs they expect the end product to fulfil. These usually bridge the gap between the high-level business requirements and more specific solution requirements. Functional requirements are detailed descriptions of what the end product must do. Non-functional requirements have to do with the design and external interfaces of the end product as well as different implementation constraints. For example requirements involving reliability, availability and maintainability are non-functional requirements. Transition requirements describe capabilities or behaviour that facilitate transition from the current state of the enterprise to the desired future state. After the transition is complete these requirements become obsolete. Examples of transition requirements include recruitment, education and migration of data from one system to another.

Gathering requirements is one of the most important tasks relating to the beginning of the project. Without them it is impossible to know whether the end product will be what it is intended to be. There are many different ways of trying to acquire the requirements. Maiden and Rugg [34] have listed twelve different methods of requirement acquisition. These are observation, unstructured and structured interviews, protocol analysis, card sorting, laddering, repertory grids, brainstorming, rapid prototyping, scenario analysis, RAD workshops and ethnographic methods.

After the requirements are gathered, it is important to document them in requirements specification. Requirements specification works as input to the design team as well as quality assurance and the reference to development manager. All in all it is the basis of communication to all parties and controls the evolution of the system [24]. Because the requirements specification has such a vital role in communication it is important that it is written using terms all parties understand. For example doctors and other medical staff use different language from developers. It is important that the requirements specification is understood by both.

Most documents created after requirements specificiation are based on it. Every feature created must be connected to at least one requirement. Tests must cover each requirement. Therefore it is vital that each requirement is traceable. This means that its origin is clear and it is referenced in documentation regarding future development of the end product. Backward traceability, to previous stage of development or all the way to the requirement's origin, depends on the requirement explicitly referencing

its source in earlier documents. Forward traceability, in all future documentation, depends upon the requirement having a unique name or a reference number. Forward traceability is especially important when the product enters the maintenance phase. When documents are modified it is vital to ensure all affected requirements are identified and updated.

It is also important to review the requirements regularly to ensure that they have not changed or new ones have not emerged. All requirements cannot be known at the beginning of the project and the ones that are known are bound to change. If changes occur the requirements specification must be updated to include the new requirements. When features are connected to requirements it is easier to see which features must be changed after a requirement has changed.

Requirements management is connected to many other aspects of project management as can be seen in Figure 2.1 on page 4. This highlights the importance of proper requirements management.

Requirement management is as important in multivendor projects as in other projects. It is especially important for the customer to know the requirements each component has for the others. For example if a high level requirement changes, multiple components' interfaces might need to be changed as a result. It is important to inform the vendors that there are new requirements for the interfaces of their product. Otherwise the changes might not get done.

2.4 Task Management

Task management is closely related to requirements management (Figure 2.1 on page 4) [30]. If a requirement is specific enough it can be a task in itself. Usually requirements are broken down to smaller bits, tasks, which are smaller and more specific than the original requirement. A task is a clearly defined entity which can be assigned to a single person [27]. The smaller a task is, the easier it is to estimate how long it takes to implement. If each task's implementation time is fairly easy to estimate it is possible to give a fairly accurate prediction on how long the software takes to complete. This prediction is rarely correct, partially because there are always tasks that take longer than estimated, and partially because requirements tend to change during the development process. [27]

The status of a task can be any of the following: ready, assigned, terminated, expired, forwarded, finished and failed [29]. Figure 2.2 shows how a task's status can change over its life cycle.

Task management covers the entire life cycle of a task. This means the planning, testing, tracking and reporting of a task. Planning includes the definition of the task, the assigning of the task to a developer, and the estimation of the time it takes to implement the task. Testing means the implementation of the task is thor-

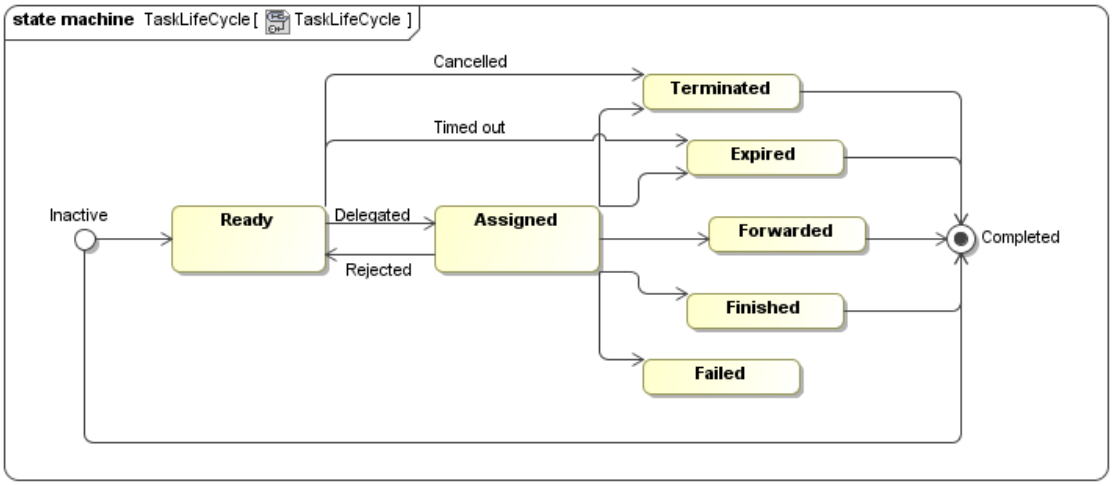

Figure 2.2: The states of a task during its life cycle. Based on [29].

oughly tested. Tracking a task means knowing when a task is implemented, when it is tested, if any bugs are found and when the possible bugs are resolved. Reporting a task includes reporting the time it took to actually implement the task. As with requirements, not all tasks are known at the beginning of a project. Task management must be flexible enough to allow changes to tasks and creation of new tasks throughout the project.

There are many ways to handle task management. A simple way is to write all tasks into a document. Either that document is updated when any changes to the tasks arise or a new version of the document is created when a change occurs. More often tasks are managed with a task managing software. There are many different kinds of task managing software systems on the market. Often project management and calendaring software provide task management software.

A task can be dependent on another task. For example a graphical user interface component needs to be created in order to see whether its components – buttons, text, etc. – are placed correctly. In multivendor projects the tasks of one vendor might be dependent on the completion of the tasks of another vendor. The vendors need to know these dependencies so they can prioritize their tasks so that no vendor has to wait for prolonged periods of time for the other vendor to complete a certain task.

2.5 Configuration Management

Configuration management's goal is to track and control the changes in the software or, in other words, to track and control the versions of each component and of the entire software [27]. From Figure 2.1 on page 4 it can be seen that version control is a part of configuration management and through version control requirements man-

agement, task management, defect management and test management also relate to configuration management. Version control systems solve the tracking and controlling of changes easily and therefore version control is often a big part of configuration management. However that is not all there is to configuration management. It also includes the practices and processes used when a component or a version is created or changed [27].

The main questions configuration management answers are "when was this task (or bug fix) implemented?" and "what chance could have caused this bug?" Usually to help answer these questions a task identifier and a version number of a component, or the entire software, are tied together. This can be done in many ways. One way is to write a comment detailing which task or tasks a change tries to implement when creating a new version of a component or software. For example "Added a drop box with which user can select month to GUI (Task #30)". Another is to add information of each version having to do with a certain task in the task managing system. For example if tasks are managed in a chart, a special cell can be created to hold the version numbers of the software that are related to the task. Some tools used to manage tasks and defects can be set up to automatically link the task and versions related to it.

Configuration management is especially important when components are used in different products. How can components be devised so that they can be used in each product? Which requirements belong to which version of each component and product? Which components and which versions of each component belong to a certain product? Configuration management is responsible for these issues.

The maintenance phase of the software's life cycle is also heavily dependent on configuration management. It is important to know which version of the software and each component in the software is being maintained so that the right changes can be made as quickly as possible and all possible dependencies to other software can be identified easily.

Pressman identifies five configuration management tasks [38]. The first one is the identification of objects in the software configuration. It is impossible to manage something if you do not know what you are managing. The second one is version control. As stated above, it is an important part of configuration management. The third task is change control. For a large software engineering project, uncontrolled change rapidly leads to chaos. Especially in larger projects changes cannot be made without good reason. When something is changed there is always a risk of unintended side effects which can cause problems. Additionally if a change is made unrecorded and later on someone else has to look for that change, it is nearly impossible. Fourth task is configuration auditing. This complements the formal technical review. The configuration audit can be done as a part of the formal technical review or separately.

The fifth and final task of configuration management is status reporting, which answers the following questions: What happened? Who did it? When did it happen? What else will be affected?

Despite the fact it is possible to identify tasks that configuration management has to do, there are no universal tips or methods that can be applied to configuration management.

2.6 Version Control

Version control is closely connected to configuration, requirements, task, test and defect managements. This is because differet versions usually implement a requirement, task or bug fix. When tests are automated or written together with the code they are also under version control. Its goal is to provide a stable development environment to the developers while keeping track of the latest version of the software. A version is a unique state of a computer software. It usually has a unique version number or version name to identify it. If numbers are used to identify a version they are usually in ascending order so it is easy to know which version is the latest. For example version 1.0 is followed by version 1.1 which is followed by 1.2 and so on. A new version is created when a change that does not cause the software to break has been implemented.

Development environment is stable when each developer can do changes to the software without having to fear that someone else is simultaneously working on the same part of the software resulting in broken software. It also means that a user can trust that the software they use will not change without them knowing about it.

As part of project management version control is used to track when a task or requirement is implemented. When a note describing what has changed is attached to each version, tracking the progress of requirements, tasks and bugs is easy. With version control it is also easier to keep track and control which features are included in a release to a customer.

Version control can be tricky in multivendor projects. If the software developed by different vendors are completely independent units the fact that the project has multiple vendors causes no problems. If there is an interface between the software developed by different vendors things get more difficult. Even if the interface can be defined perfectly in the beginning, chances are it will have to be updated later on. When the interface changes it might be difficult to determine who should change their interface to work with the other vendor's interface, and which versions of each software are compatible with each other.

2.7 Test Management

Testing is an essential part of the quality assurance of a product. How can you be sure that the product functions in the way it is expected to function and not the way it is not meant to function, if the functionalities are not tested?

Software testing is often closely related to the verification and validation of software. IEEE Standard Glossary of Software Engineering Terminology defines verification as "The process of evaluating a system or component to determine whether the products of a given development phase satisfy the conditions imposed at the start of that phase." In layman's terms it means "is the software built right?" Validation on the other hand is "[t]he process of evaluating a system or component during or at the end of the development process to determine whether it satisfies specified requirements." Or in simpler terms "has the right software been built?" [16]

Since the verification of a software system means checking that the software is built the way it was supposed to be built, test management relates to requirements and, through that, to task management as can be seen in Figure 2.1 on page 4. If tests are automated or otherwise written together with the software, test management relates to version control. In the development phase of the software defects are most often found through testing, and thus test management relates also to defect management.

There are many different aspects to software testing. For example code reviews and inspections fall under the definition of software testing due to their close relation to verification and validation. Despite these being important methods of finding defects and making sure the quality of the program is not compromised, test management focuses more on tests that are conducted by running the software.

Figure 2.3 depicts a simplified example of a test. In the test the software is given input X, which causes the output Y. Input and output in this case can mean various things. Input can be anything from a string being entered into a text box to a physical lever being pushed in a machine. Output can in turn be anything from a computer screen updating to a light turning on to a mechanical arm moving.

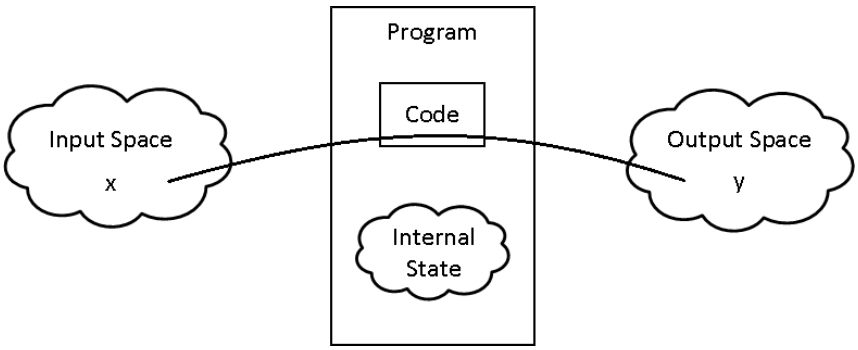

Figure 2.3: Software testing. Adapted from [27].

The result of the test depends on the output and the internal state of the software. The internal state means for example the values of internal variables and registers. In order to the test to have succeeded the output Y needs to be correct and the internal state of the software to have changed correctly. This requires the tester to have an idea of what to expect for the output and the internal state at the end. This is why all tests should be based on the specification of the software.

Not everything can be tested. Testing every single input and input combination is almost always impossible. It is also often unnecessary. If a program adds numbers together and it is tested that it correctly sums 2 and 2 together, it is likely that it will also sum up 2 and 3 correctly. However it is unclear how the program responds if one of the inputs is a negative number or empty. Hence it is often more important to focus on boundary, edge and corner cases rather than testing everything [26, 33, 22, 35, 40].

Non-functional aspects of quality are also difficult to test. Non-functionality focuses on what the program is supposed to *be* instead of what it is supposed to *do*. These include usability, scalability, performance, compatibility and reliability. These are highly subjective and it is therefore hard to determine what level of usability, for example, is sufficient.

Tests need to be managed just as requirements. In fact test and requirements go often hand in hand – each requirement needs to be verified by testing. Tests can be designed as soon as the requirements have been gathered but since requirements are likely to change, tests might change as well. Completely new requirements and features cause changes to tests as well. Also many tests are often tied to the way the software is programmed. If a defect is found and fixed so that the code is changed some test might need updating.

There are many ways of handling test management. ISO 9000-3 [17] requires a test plan and test report on both system and integration testing. Due to the standard, test documents are widely used. There are also many programs that help with test management such as TestLink and IBM Rational Quality Manager [15, 12].

In multivendor projects each vendor usually tests their own components. However the integration of the different components needs to be tested as well. Usually this falls on the customer – either they have an in-house team handling integration testing or they outsource this as well. As the customer tests the integration they should also test the functionality of the components more broadly since the project is finished when the customer signs it off approving it.

2.8 Defect Management

A defect is a behaviour of the program that is different from the behaviour described in the specification of the program [27]. This means that if there is no specification there cannot be any defects. Sometimes defects reveal hidden requirements or ex-

pectations that the customer has that were not documented in the beginning of the project.

This means defect management is closely related to requirements management. It can also be seen in Figure 2.1 on page 4 that it is related to task and test managements and version control, and through it to configuration management. Defects are usually found via testing and fixing a defect is a task. When a task is implemented there is a trace of it in version control. Thus when fixing a defect is a task, there is a trace of it in version control after the fix.

Although a different term can be used depending on the kind of defect and the stage in which it is detected, they are often used as synonyms to defect [19]. These words include error, mistake, bug, fault and failure.

When managing defects it is important to consider at least the severity and frequency of the defect. For example the software crashing is a very severe problem. However if the likelihood of the software crashing is very low there is next to no reason to worry about it. Then again if a button is located so that a user has to look for it every time they use the software, the defect is more important despite the severity of the bug being low (the button exists, so its function can be run). Often there are other factors chosen by the company that affect the assessment of the defect in addition to the severity and frequency of the defect [36].

There are many tools available to help with defect management. Sometimes these tools are expanded to cover also other areas of project management. For example JIRA is used for requirement and task management as well as defect management [9].

In multivendor projects defect management can be difficult. It is not always clear whose component is responsible for the defect. For example a defect in component A can make component B behave in undesired way. The defect would then seem to be the responsibility of the makers of component B, though in fact they cannot do anything about it. Even if the makers of component B identify the origin of the defect as component A it might be difficult for them to convince the customer and the makers of component A that the defect is A's responsibility. This is made easier if all vendors share the same defect managing tool where they can discuss and reassign defects between each other.

2.9 Risk Management

Risk management is a more isolated aspect of project management than the others as can be seen in Figure 2.1 on page 4. It consists of outlining the risks and preparing for them. The outlining of risks includes identifying probable risks, analysing these risks and estimating the probability of these risks actually manifesting. The preparation for the risks means trying to minimize the propablity each risk will actually happen

Table 2.1: The Top 10 Software Risk Items according to Boehm [18].

Risk item	Risk-management technique
Personnel shortfalls	Staffing with top talent, job matching, team building, key personnel agreements, cross training.
Unrealistic schedules and budgets	Detailed multisource cost and schedule estimation, design to cost, incremental development, software reuse, requirements scrubbing.
Developing the wrong functions and properties	Organization analysis, mission analysis, operations-concept formulation, user surveys and user participation, prototyping, early users' manuals, off-nominal performance analysis, quality-factor analysis.
Developing the wrong user interface	Prototyping, scenarios, task analysis, user participation.
Gold-plating	Requirements scrubbing, prototyping, cost-benefit analysis, designing to cost.
Continuing stream of requirements changes	High change threshold, information hiding, incremental development (deferring changes to later increments).
Shortfalls in externally furnished components	Benchmarking, inspections, reference checking, compatibility analysis.
Shortfalls in extemally performed tasks	Reference checking, preaward audits, award-fee contracts, competitive design or prototyping, team-building.
Real-time performance shortfalls	Simulation, benchmarking, modeling, prototyping, instrumentation, tuning.
Straining computer-science capabilities	Technical analysis, cost-benefit analysis, prototyping, reference checking.

and making a "plan B" in case that risk actually is realized. [27]

There are different checklists to help with this. The most famous is probably written by Boehm [18]. It lists the top 10 risks of software projects and techniques with which to manage those risks. This checklist is shown in Table 2.1.

Usually the highest risks have nothing to do with technology but are more concerned about human errors, such as problems in project planning, tracking and organization. Often overconfidence turns out to be the greatest realized risk.

In smaller projects project manager is usually in charge of risk management. This is then one of the most important tasks of the project manager. In larger projects a more formal risk management plan can be designed and periodically overviewed by upper management.

In multivendor projects the greatest risk is usually the changing of the requirements. Since multiple vendors work on the same project one change in the requirements can result in changes in many components and thus interfaces. If an interface

needs to be changed, it might be difficult to decide whose responsibility it is to change the interface.

3. VERSION CONTROL REVISITED

Version control is an important part of software development. It is the practice of tracking and controlling the changes in the source code. This can be done purely by the developer, without any version control software, or developers although there is a multitude of software to help with this such as Subversion (SVN), Git and Concurrent Versions System (CVS) [14, 4, 2]. Sometimes version control is used to manage documentation as well. There are document management systems which specialize on handling the version control of documents.

Nowadays software is usually developed in teams. In them developers work on different updates simultaneously. This means each developer has a version of the software they are working on, which is slightly different from the other developers' versions. When these versions are merged with the official version of the software, it is important to know if someone else has already made changes to it. It is also important that two developers cannot add changes to the official version of the software at precisely the same time.

There are many reasons for tracking and controlling changes. It is rare that a software release is completely without errors. It is important to be able to find a version that the bug first appeared and therefore the changes that might have caused it. When a version of the software is released to the customers it does not necessarily contain all the updates that have been done before the release date. Version control helps determine when each update was added and thus which version to be released. And sometimes it is necessary to develop multiple versions or "branches" of the software simultaneously. This is the case for example if the most up to date version only supports a later operating system than an important customer has and there are bugs to be fixed in both.

3.1 Structure

There is a certain structure in all version control systems. There is at least one *repository* which is where files' current and historical data are stored. When a developer wants to work on the software he *checks out* a copy from the repository to his computer. This copy is called a *working copy* since all the work is done in this copy. When the developer wants to add the changes he made to the repository he *checks in* or *commits* the changes he made. When two different versions are to be developed at

the same time a *branch* is made. This means that the files are copied made and from there on it is possible to make certain changes to one copy – or branch – and other changes to the other. If the changes in two branches are needed in the software, the branches are *merged* together. There is a *conflict* if both branches contain different changes to a particular part of the software. If there is need to take a snapshot of the software, i.e. for a release, a *tag* is used to accomplish this.

Version control can be easier to understand if it is considered to be an acyclic graph or a tree-like graph where the branches merge back into the tree. The simplest case is where no branches are used and all revisions belong to the *trunk*. The graph in this case is just a line. The latest version in this line is called the *head*. When branches are used the graph starts to look more like a directed tree. However, when a branch is merged back to the trunk the graph turns into a directed acyclic graph.

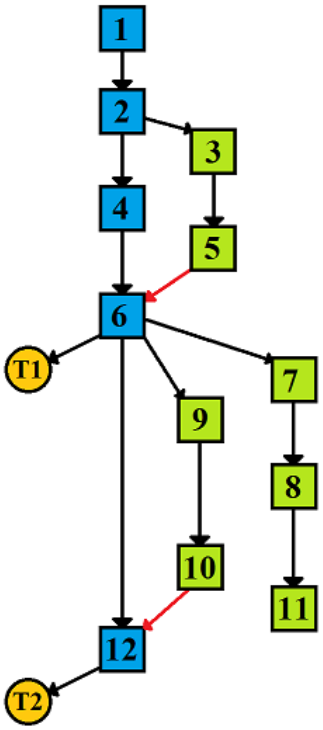

Figure 3.1: Version control depicted as a graph.

Figure 3.1 shows an example of a version control graph. The trunk is represented by the blue squares. The green squares are versions in a branch. The merging of a branch back to trunk is signified by a red arrow. Tags are represented by the yellow circles. From the picture we can see that it is possible to develop many branches simultaneously. Sometimes branches are discontinued and never merged back to trunk. The branch containing versions seven, eight and eleven is one such branch.

3.2 Source-Management Models

Version control is traditionally implemented so that there is a central server which has the official version of the software and the developers check out and commit their changes on this server. When there are multiple people working on the same project certain error scenarios can occur.

3.2.1 Atomic Operations

An operation is atomic if it appears to take effect in a single instance [31]. In other words an atomic operation either succeeds completely or does not happen at all – there is no state between.

Let us assume two programmers Tom and Jerry are working on the same project. Tom has finished some changes to the project and commits the files to server, but the connection is lost in between his computer and the server before all the changes are written to the server. Jerry checks out the latest version of the project to his computer only to find that the project does not compile anymore because only some of the changes Tom had made it to the official version of the project on the server.

To avoid this most, though not all version control systems have implemented commits as atomic operations [1]. In the example above Tom would have seen an error message explaining that he could not commit his changes because the connection was lost and Jerry would have gotten the previous version of the project.

3.2.2 File Locking

Lack of atomic commits is not the only problem Tom and Jerry could face. It is more likely than not that at some point Tom and Jerry would both edit the same file without knowing of each other. Jerry commits his changed version first. When Tom tries to commit his version, the server tells him that the file has changed since he last checked it out and his changes cannot be committed. He then has to check out the new version of the file and add his changes to the new file and hope Jerry has not changed the file again before he has time to commit the changes.

This can be solved in two ways. The first is to lock the file while someone is making changes so others cannot edit it at the same time [28]. With file locking Jerry would do his changes and Tom could only read the file and plan the changes he would make while waiting that Jerry would commit his changes. This works well when Jerry does not edit the file for long. But if Jerry forgets to commit his changes and thus release the lock on the file before going on a holiday, Tom cannot do anything but wait until Jerry gets back. He could bypass the revision control and edit the files locally but this would make it difficult to merge Tom and Jerry's edited files later on.

3.2.3 Merging

Another way to solve the problem above is to merge the files. Merging combines changes from multiple source branches into a single target branch, initiating a conflict resolution process if changes are incompatible [37]. If merging was available in the example Jerry would commit his changes without any problems. If Tom had edited a different part of the text file he could also commit his changes without any problems. If however he had edited the same part as Jerry the version control system would inform him of this and he would have to merge the versions by hand.

3.3 Different Implementations

There are multiple ways in which version control can be arranged. These are local only version control, centralized version control, distributed version control and completely synchronized version control.

3.3.1 Version Control By Hand

The most basic way of handling version control is to do it all by hand. The simplest way to do this is to copy the work to a different directory. If the directories are time-stamped this makes it easier to keep track of which version is in which directory. It is a popular way of arranging version control manually but it is also very prone to errors. It is easy for the user to forget which directory they are in which can lead to them working on the wrong copy or copying over the wrong files.

3.3.2 Local Only Version Control

Local only version control is the simplest form of version control. It tracks the changes to the files that are stored on a single computer. This means that work for the project can only be done on one particular computer. The first version control software was released in the 1970's [39].

To get rid of the human errors ever present when handling version control by hand, the first version control systems or version control systems for short were created. The first ones focused on version control on one computer. They had a simple database that kept track of all the changes to files. Figure 3.2 depicts the idea of local only version control systems.

An example of local only version control system is RCS (Revision Control System) which is still distributed with many computers today [7]. For example when you install the Developer Tools to Mac OS X, RCS will be included [11].

Local only version control in included in more modern version control systems as well as the more complex version control they are designed for. Therefore systems

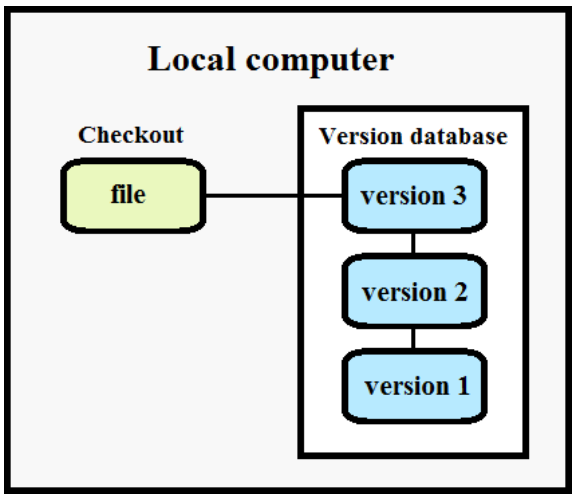

Figure 3.2: A simplified picutre of local only version control [23].

created for purely local only version control are not very widely used anymore.

3.3.3 Centralized Version Control

When multiple developers work on the same project it is useful that they can work on different computers. For this purpose centralized version control systems were created in the 1970's and 1980's. In a centralized version control system a repository which contains all the files resides on a single server. All developers check out a copy of the version on the server to their computers to work on. This requires the developers to have network access to the server. For years this has been the standard for version control. The diagram in Figure 3.3 describes centralized version control systems.

Concurrent Versions system (CVS) and Subversion (SVN) are probably the most popular centralized version control systems [14, 2]. Especially Subversion has been very successful and seems to be the most widely spread version control system out there today [14].

Centralized version control systems have many advantages when compared to a local only version control system. For example developers have some kind of insight as to what the others are working on. The administration of a centralized version control system is a lot simpler than that of multiple local only version control systems. centralized version control systems also allow the administrators control over who can do what.

There are also downsides. The greatest downside is that the repository is on a single server. If that server goes down, nobody can collaborate or save the changes they have made until the server is back online. If the hard disk of the server is corrupted and no backups are made, all the work will be lost. Only the snapshots each developer might have on their own computer are left. This same problem exists

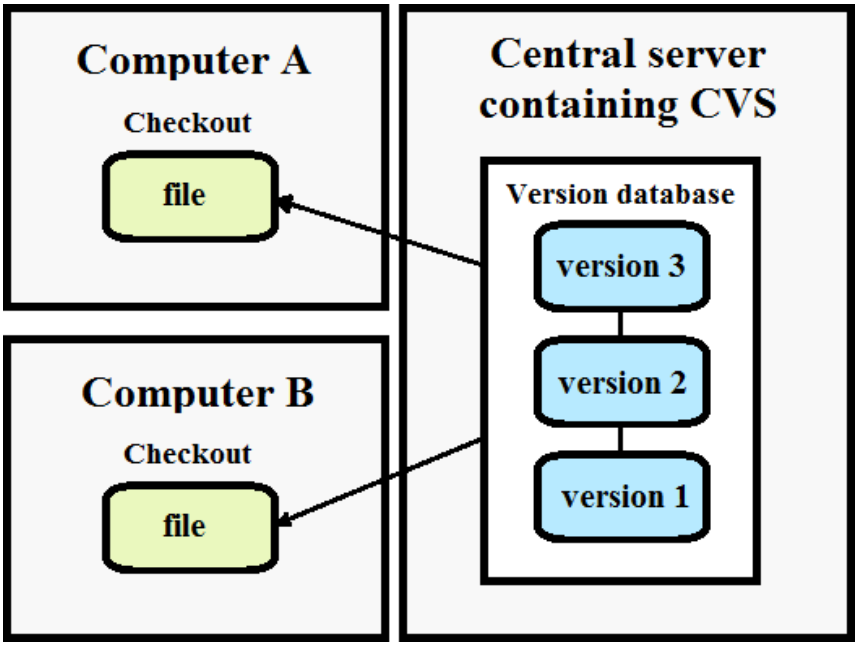

Figure 3.3: Centralized version control [23]

.

also for local only version control systems – if there is only one place where data is saved, the risk of losing it all is always there.

3.3.4 Distributed Version Control

In a distributed version control system, created in the 2000's, there is no central server that everyone has to interact with. Each developer mirrors the entire repository on their computer. This makes it possible to work with a version control system even when the developer's computer is not online. If one of the computers or servers containing a repository goes down, it is easy to recover even if backups are not fastidiously made. Figure 3.4 shows the idea of distributed version control systems.

Another upside to distributed version control systems is that it is possible to use several "master" repositories whereas centralized version control systems allow only one. This makes it is easy to collaborate with multiple sets of people simultaneously.

Git and Mercurial are some of the most popular distributed version control systems at the moment [4, 10]. Neither is as popular as the most popular centralized version control systems are, but it is commonly accepted that it is likely distributed version control systems will become more popular than centralized version control systems in time.

There are multiple ways in which a distributed version control system can be set up. The homepage for Git explains three ways to arrange the repositories and workflows: centralized workflow, integration manager workflow, and dictator and lieutenants workflow [4]. However it is possible to set up many different kinds of

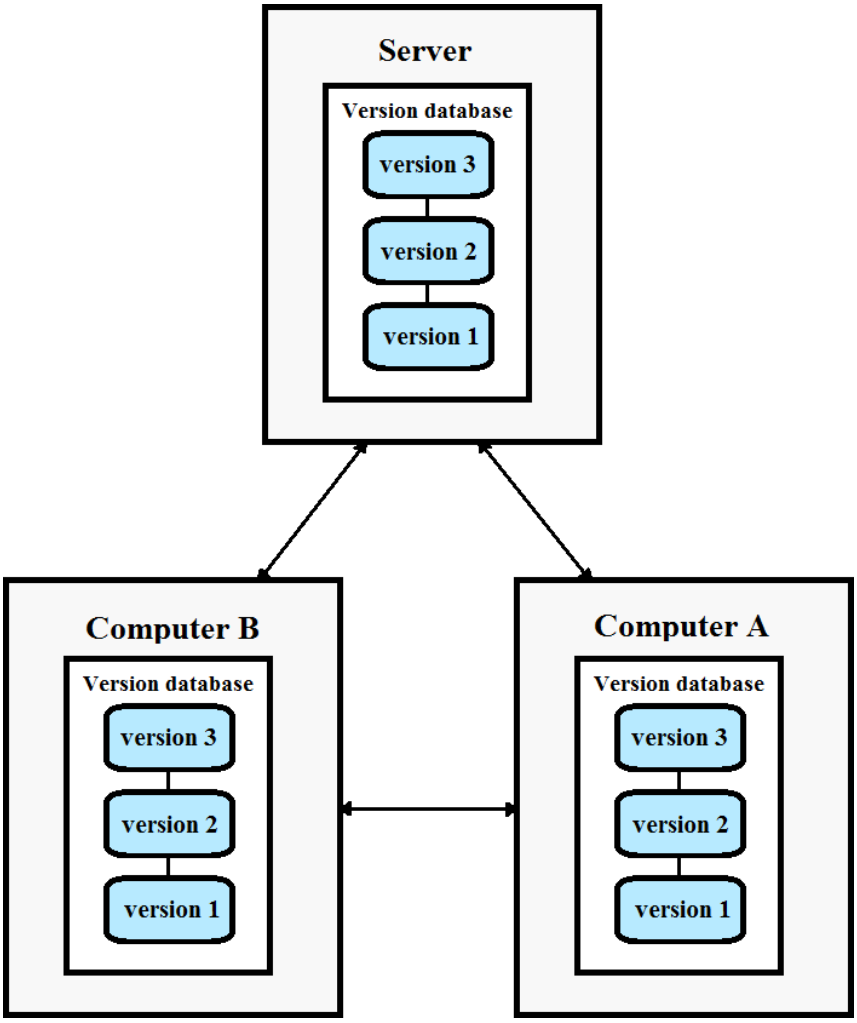

Figure 3.4: Distributed version control [23].

other workflows as well.

Centralized Workflow. A centralized workflow is the same workflow as in centralized version control systems: there is one server with the official repository. The developers pull from and push to it the changes they have made on their computers. This is the most common workflow and popular especially among developers who are used to using a centralized version control systems. Centralized workflow is shown in Figure 3.5. [5]

Integration Manager Workflow. Another common workflow includes an integration manager or a single person who has the right to commit to a "blessed" repository. The developers can clone from that repository, push to their own repositories and ask the integration manager to pull the changes they have made. This sort of workflow is commonly used in open source projects. Figure 3.6 shows how the integration manager workflow works. [5]

Dictator and Lieutenants Workflow. The dictator and lieutenants workflow is similar to integration management workflow but it is in a larger scale. Each lieu-

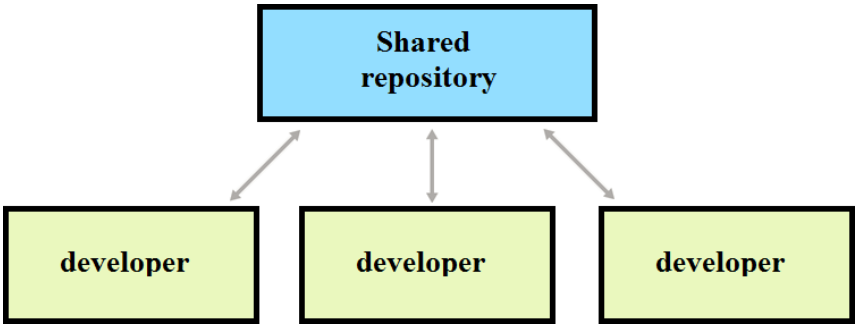

Figure 3.5: Centralized workflow [5].

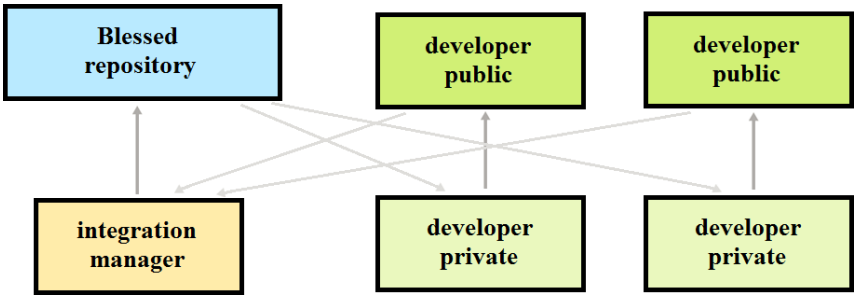

Figure 3.6: Integration manager workflow [5].

tenant is responsible for a specific subsystem and act as integration managers for those subsystems. The dictator is another integrator but they can only pull changes from the lieutenants. The dictator can then push to the "blessed" repository which the developers (under the lieutenants) can clone again. Dictator and lieutenants workflow is depicted in Figure 3.7. [5]

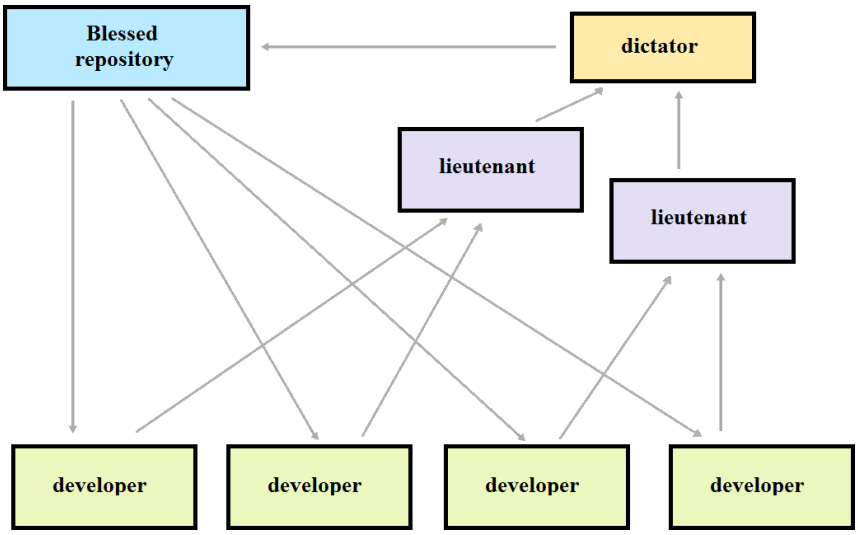

Figure 3.7: Dictator and lieutenants workflow [5].

A famous example of this workflow is the Linux kernel [5]. It has been divided into sections for which many developers contribute to. A lieutenant picks the changes he

deems are fit to be passed up and the dictator reviews the selected changes before pushing them to the "blessed" repository.

3.3.5 Simultaneous file editing with version control

Google Drive allows users to make changes to a file in a browser at the same time. Not only that, but it also enables users to see the changes the others are making as they are making them. It also keeps track of the changes made and makes it possible to look at and bring back older versions much like official version control tools. The documentation states: "While it may not work exactly like a track changes tool, the revision history tool lets you view and revert to earlier versions of your document, spreadsheet, presentation, or drawing and see which collaborators made edits to any of these versions" [13]. Figure 3.8 shows an example of a revision history in a Google Drive file. Different versions can be browsed and restored. Restoring a revision moves it to the top of the list leaving the changes made to later versions untouched and browsable.

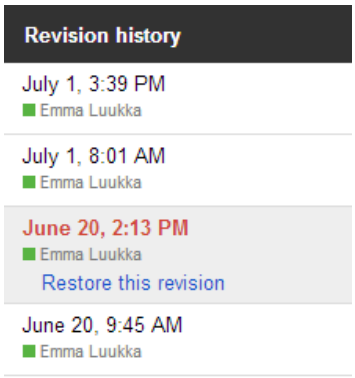

Figure 3.8: An example of Google Drive's version history.

It is also possible to download Google Drive application to your computer or phone and manage your files that way. The Google Drive directory on your computer is synchronized with the web page. This means the files in the Google Drive directory can be seen on any computer with internet access and that they are also under Google Drive's version history. However the files are not stored on your computer. Instead your computer has a link to the file which will can be opened in the browser for viewing and editing.

Using Google Drive in a programming project is not a feasible idea even if one thought that seeing changes another programmer is making, as they are making it, is useful. Google Drive allows you to work on text documents, presentations, spreadsheets, forms or drawings. These are all similar to what Microsoft Office offers. Therefore the advantages of using Google Drive are lost when working on code files – the file types listed are not the the most optimal for writing code: each of them

add characters in the form of automatic formatting and none of them have support for syntax highlighting which many programmers today are used to. Working on a code file, that is synchronized to Google Drive, in any other program takes away the thing that differentiates Google Drive from other available version control systems – seeing the changes as others are making them.

There are also issues with the reliability of Google Drive's version control system. If the space allocated to the owner of the file is running tight, Google Drive will combine changes done close enough together as one version. This is reasonable, since Google Drive saves changes often – if the user works on a document for an hour, Google Drive will have multiple versions of it saved by the end of that hour. Initially Google Drive groups the changes together as one version but, if the user chooses to, they can see each version Google Drive originally saved. If space is running tight and the versions are old enough Google Drive will group them together as one change and delete the subversions within that version. This is probably what the user would want anyway. However if space is really running out for the user, Google Drive will delete oldest versions of documents. This might be okay, but it is not what one would expect or be pleased with in a version control system.

There have been attempts at using Google Drive as source control. So far it has not been successful [8].

4. MULTIVENDOR PROJECTS IN ATOSTEK OY

Practically all projects currently worked on at Atostek Oy are multivendor projects. Often the other vendor or vendors are working in-house for the customer. This blurs the concept slightly, but there are still multiple independent teams working on components sharing interfaces.

There are no official procedures when it comes to project management within Atostek Oy, so in order to find out the current state of affairs, a study was conducted. A face to face interview was selected as the method of information collection due to the hectic schedules of the project managers who were interviewed, and the small number of interviewees (seven). A questionnaire would have been easy to push back and forget – an interview forced the interviewees to concentrate. When there are only a few interviewees it is sensible to acquire qualitative information rather than quantitative. Because it was expected that each project would be handled slightly differently from the others it was difficult to set out to collect quantitative data. It was easier to plan a semi-structured interview than for example a questionnaire.

4.1 Project Management in Multivendor Projects in Atostek Oy

In the interviews it became clear that in each project some aspects of project management were handled more systematically – that is, there is a concious plan to handle that aspect – than others. Two factors seemed to contribute to this. First, things that were considered very important and difficult to handle by humans alone, such as version control, were often handled systematically with a tool. Also the size of the project, the level of experience of the project managers, the level of project managers' trust in their own abilities and memory, and the perceived signifigance of an aspect of project management played a part in how systematically a given part of project management was handled. If the project manager is experienced and confident and the aspect of project management is viewed as not important, the aspect is more likely to be handled unconciously on the side of other project management tasks. However if the project manager is new and not very confident, they are more likely to want to ensure each aspect of project management is handled and therefore

conciously make plans and systems for each aspect of project management. The level of systematic handling of aspects of project management is shown in Figure 4.1.

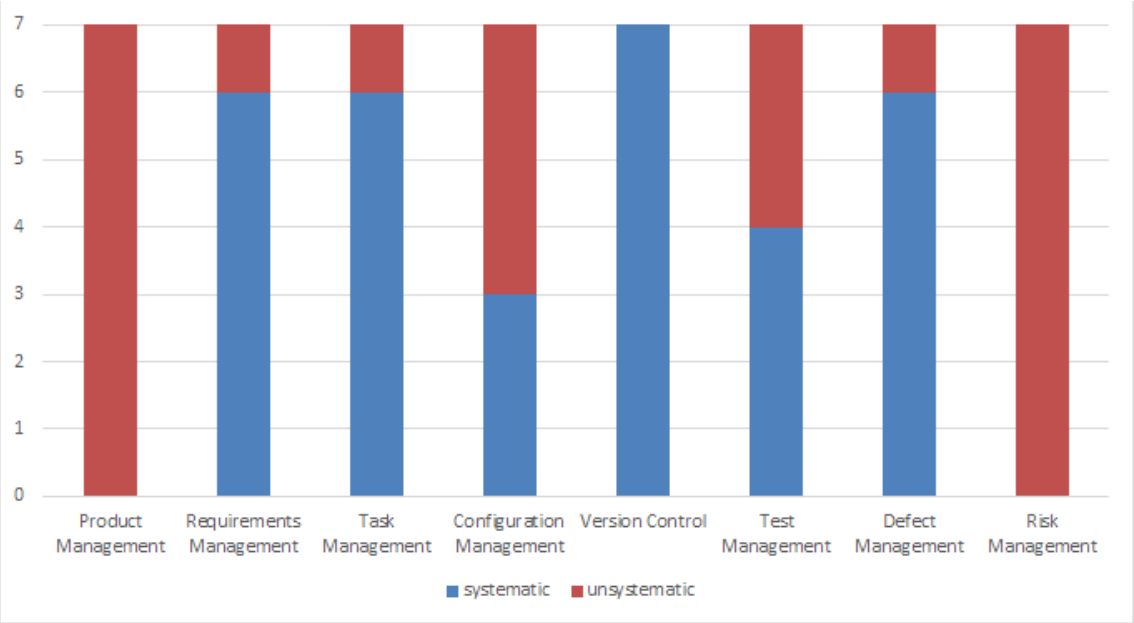

Figure 4.1: How systematically different aspects of project management are handled in Atostek Oy.

It is also interesting to note which tools are used to handle each ascpect of project management. Some tools are used in handling multiple things whereas in some cases no tool is used. The tools used are shown in Figure 4.2.

Product Management in Multivendor Projects in Atostek Oy. There is no clear consensus in Atostek Oy about what product management is. As per Figures 4.1 and 4.2 product management is not actively considered in any of the projects in Atostek Oy. The projects that could identify product management stated that it was handled by the customer. The other project managers conceded product management was not actively considered in the projects. Out of the project managers that were interviewed two admitted they did not have a clear idea of what product management is, four considered it to be synonymous to configuration management and one thought it had more to do with the entire life cycle of the product and therefore not really relevant to a bespoke company such as Atostek Oy.

Requirements Management in Multivendor Projects in Atostek Oy. Requirements are mostly managed in some formal way as can be seen in Figure 4.1. In only one instance requirements are not really managed at all – they are conveyed via emails, various documents or phone conversations. In four cases a tool is used. However different tools are used, and in each case the tool is originally meant for bug tracking. Often the same tool is used for requirement, task and defect managements. Three projects stated all vendors working on the project shared the

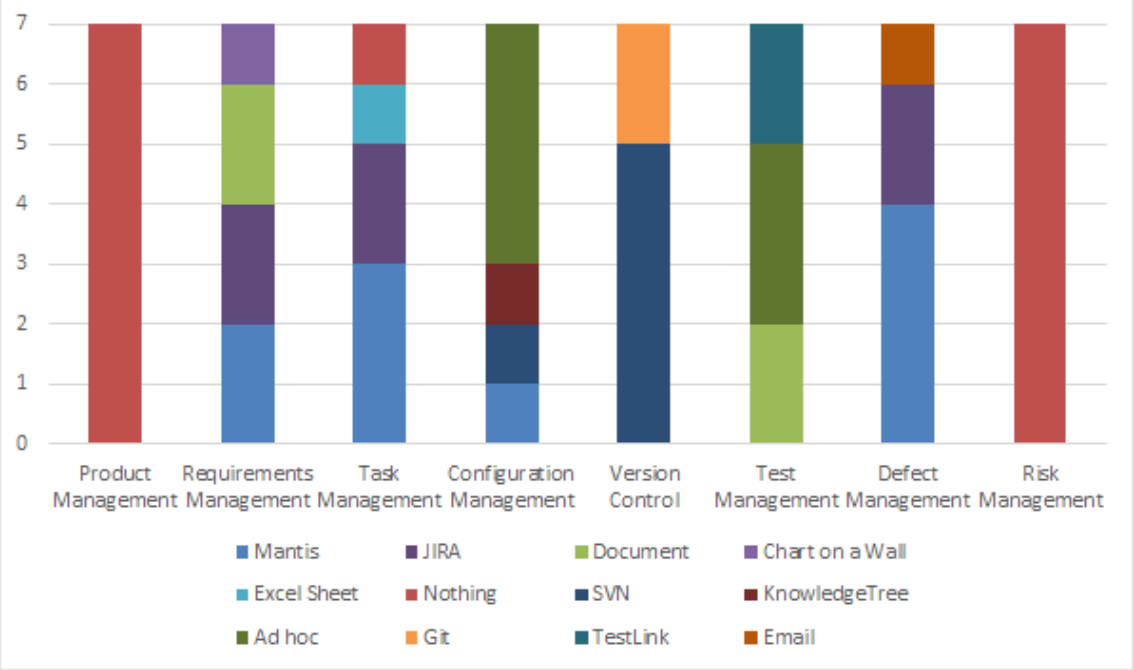

Figure 4.2: The tools used in handling different aspects of project management in Atostek Oy.

same tool for these tasks. These projects also said communication and delegation between vendors worked well. In three cases some other method was chosen to manage requirements. In two of those cases requirements specification document is used as the basis of requirements management. In one of the cases a chart is created based on the requirements specification and kept on the wall for all developers to see. Though only one is solely dependent on meetings, emails and phone calls, these are important in all projects in keeping the requirement management tools up to date. The different tools used in requirements management in Atostek Oy can be seen in Figure 4.2.

Task Management in Multivendor Projects in Atostek Oy. All projects but two use task management software to manage tasks internally as can be seen in Figure 4.2. Of the remaining projects one has assigned and managed tasks by word of mouth, although the project manager is looking for a better way of doing this and distributing information between developers. The other an Excel sheet designed for task management. There is more diversity with regard to making task management transparent to the customer. In three cases the same tool used for requirement, defect and task management is shared with the customer so they can see the progress made. In one instance a document containing tasks is updated and released together with the latest version to the customer. In the rest of the cases the tasks are managed purely internally without the customer knowing the status of the tasks.

Configuration Management in Multivendor Projects in Atostek Oy.

There is no general way to handle configuration management in Atostek Oy. In one project all vendors upload their components to a common database (KnowledgeTree) where the others can easily get the latest version of each component. In two cases the developers in Atostek Oy compile all parts of the final program together. In this case other vendors deliver their components (often through the customer) to Atostek Oy where the final product is compiled. The customer also informs the developers in Atostek Oy, which versions of each component is to be used in each release. In the other projects configuration management is handled nearly completely passively after the version control is set up. The usage of branches, when a bigger change is to be made to a project, and tags, when a release is made, is common practice in Atostek Oy though no official procedure is in place. The different methods used to manage configurations in Atostek Oy and their distribution can be seen in Figure 4.2.

Version Control in Multivendor Projects in Atostek Oy. Centralized version control systems and especially Subversion is the de facto version control system in Atostek Oy. There is, however, a growing trend of using distributed version control systems. Currently two projects use Git while five use Subversion as per Figure 4.2. Version control is mostly handled separately by each vendor in the projects Atostek Oy participates in. Six out of seven projects do not have access to the other vendors' code and instead design the interfaces between components with the other vendors. One project uses a shared version control with other vendors. Two other project managers are hoping to migrate to a version control system that is shared by all vendors. One of these projects already uses Git within Atostek Oy – something they wish will help persuade the customer to switch to a shared version control system. One project is in the process of changing their version control system to a shared system – using Mercurial – due to the customer's wishes.

Test Management in Multivendor Projects in Atostek Oy. Test management and testing in general seems to divide the projects in Atostek Oy into two groups: the ones that do it formally and the ones who do not. That does not go to say that some projects do not test their programs at all. All developers test the changes they make into their projects. Some developers test more thoroughly than others – there are no common practices to testing set by the management of Atostek Oy. Two projects use a testing tool to manage their tests. One has used it for a long time while the other is such a new project that they are only planning on using it. In two other cases a word document is used to prove to the customer at the end of the project that testing has been conducted. Three projects conduct test management in a purely ad hoc manner without any plans or predefined practices. In two of these cases the customer has a team that does testing so the development team feels their testing can be more informal. In the remaining case a specific plan for more vigorous

testing is drafted together with the customer. Two projects dreamed of automated tests, but had not taken steps toward it yet. The different methods used to manage tests in Atostek Oy and their distribution can be seen in Figure 4.2.

Defect Management in Multivendor Projects in Atostek Oy. Almost all projects currently in progress in Atostek Oy use a bug tracker to manage defects. In some cases all vendors use the same bug tracker, which makes it easier to reassign defects if it turns out the defect is actually in a component another vendor has implemented. In two cases defects were emailed to the project manager in Atostek Oy, who would add the defects to a bug tracker used internally in Atostek Oy. In one project a bug tracker was not used during the interview but its use was planned for the future. In one case defects were managed purely via email. The distribution of different defect management methods used in Atostek Oy are shown in Figure 4.2.

Risk Management in Multivendor Projects in Atostek Oy. Risk management is not formally planned in any project by Atostek Oy. Each interviewee seemed surprised to be asked about risk management. After a moment's consideration each one came up with passive ways in which risks are managed. In a few projects knowledge is actively distributed between developers. In one case the customer required this and allocated time for it. Some project managers feel risk management is removed from them by the biweekly meeting in which the upper management decides how different resources and developers are distributed between projects. The lack of risk management in Atostek Oy can be seen in both Figures 4.1 and 4.2.

4.2 More About Version Control in Multivendor Projects in Atostek Oy

There are seven multivendor projects currently under way in Atostek Oy. One of them uses a distributed version control system (Git) in collaboration with the other vendors involved. All others have separate internal version controls with various ways of compiling the final product for the customer. Out of the six projects which do not use distributed version control among different vendors one project uses Git in the hope that it is the first step toward distributed practice. All other projects use Subversion.

According to the interviews centralized version control and more specifically Subversion is the de facto version control system in Atostek Oy. It has been used in most projects and therefore it is easy to start using in new projects as well. If there are specific circumstances making a distributed version control system more desirable, it is used instead. Reasons to use a distributed version control system range from the inherent need to share source code with other vendors to the customer's demands.

The customer's demands have also stopped a distributed version control system from being used despite the projects inherent need to share source code between vendors. In this case the customer was used to working as the medium between the vendors. The customer liked this position as it meant they always knew what each vendor was doing or what they needed. They feel using a distributed version control system would diminish the need for communication resulting in the customer knowing less of the progress made by the vendors.

Using tags and branches is as integrated a process in Atostek Oy as using Subversion as the version control system. There are no company wide rules or guidelines relating to this but all projects use branches and tags similarly to each other.

The methods of delivering products to the customer vary greatly. In three projects Atostek Oy combines multiple vendors' components into a product before releasing it to the customer. In two cases, each vendor integrates and tests their components before releasing them to the other vendors and the customer combines the components to a product themselves. The project using distributed version control among all vendors delivers sources directly to the customer. In one case, each vendor delivers separate sources to the customer. This is possible because each vendor creates independent software which collaborates with the others via different interfaces.

4.3 Project Practices Checklist

As can be seen from the previous sections, there are no official procedures or guide lines at Atostek Oy in regard of general project management. This section is designed act as a checklist for project managers. The suggestions and lists here are not meant to be exhaustive, but they are meant to help them to recognize what needs to be taken into consideration when managing a project and to make informed decisions on the different aspects of project management.

4.3.1 Product Management

As Atostek Oy is a bespoke company, Atostek Oy's project managers need not focus on product management. This is somewhat true: once a deal is made with a customer, there is no need to "champion" a product anymore – it is already in production and its marketing is already done and the representative of the customer is responsible of making sure the final product is what they want it to be.

However, the project manager can support the representative of the customer. This can be done for example by focusing on understanding **what** needs to be done (as opposed to how it should be done) and gathering feedback from the representative of the customer as often as possible. In a multivendor project it is also important to know how the different functions and responsibilities are divided between vendors.

The information relating to what needs to be done as well as feedback can be difficult to document in a set way. However this information is crucial to keep constantly in mind in order to make the right product. The way it is documented does not matter but it is important to have it documented in some way – a document, free form notes, a poster, etc.

4.3.2 Requirements Management

The importance of requirements management is internalized by the project managers at Atostek Oy. It is always handled in a more or less formal way. Here are listed some ways in which requirements management can be handled. Which ever way is used, it is important to remember that requirements are likely to change and hidden requirements are very likely to pop up during the project.

A requirements document A document can be created in many ways. It can be *a traditional document* with the requirements written out or described in table. The requirements can also be drawn or printed on *a poster* as a table or more visual way. This way the requirements are always on display.

A requirements management tool There are multiple tools that can be used in requirements management. Most of them are designed to be used in other aspects of project management as well – most often task and defect management. Jira and Mantis are examples of these tools.

Notes, email, and memory In smaller and shorter projects it might be sufficient to rely purely on one's own notes, email and sometimes memory for requirements management. It is however not recommended because it is easier to lose and forget requirements when they are not managed in a single place.

4.3.3 Task Management

Task management is one of the aspects of project management that is given the most thought to in Atostek Oy and that project management knows the most about. The tools for task management are very similar to requirements management, as can be seen from the list below.

A task document This document can be done in various ways such as the requirements document. However there are likely to be much more tasks and they are more short lived than requirements so keeping *a document*, *a chart*, or *a poster* up to date could be tedious.

A tool Just like for requirements management, there are many tools designed to help handle task management. It is possible to use the tools for more than just task management. These tools include JIRA, Mantis, and Redmine.

Notes, email, and memory In very small and short projects tasks can maybe be managed by relying on one's own notes, email, and memory. However this is very risky.

4.3.4 Configuration Management

At Atostek Oy configuration management is often synonymous to version control, which is handled very thoroughly in each project. However there is more to it than that.

When releases always contain only the latest versions of each component and only the latest release is used by the customer, it is easy to make and maintain the releases. However there are cases where a release is comprised of different versions of modules – one module's latest version is included in the release while another module's older version is required. In these cases it is usual for different versions of the software to be used at the same time. This means that the versions of each module in each release needs to be known at the time when the release is made and throughout the lifecycle of the software for maintenance purposes. This can be done in many ways:

Version control tool Version control systems allow versions to be tagged. This is a good way to label the versions of different modules which belong to a release.

Document One simple way to keep track of the contents of each release is to write it in a document. Whether one document holds information for all releases or a document is created for each release, the information is easily accessible.

Notes, email, and memory Information is often conveyed via email. Single emails will easily get lost amid all other emails, which makes email an not a very good place to store information. The same applies to notes. Using a combination of the two, and relying on memory, makes things even more complicated and prone to mistakes since the information is stored in different places.

4.3.5 Version Control

Setting up a version control mechanism for each project is an ingrained process in Atostek Oy. While centralized version control systems, such as Subversion, are most

common, distributed version control systems, such as Git and Mercurial, are considered more and more. While centralized version control systems are most familiar to all employees, distributed version control systems offer many good features such as different workflow options and the possibility to work offline.

Version control can be handled in multiple different ways:

Locally without a version control tool First of all locally handled version control only works if there is only one developer working on a project. Local version control also requires regularly scheduled backups. If a version control tool is not used the version control is handled completely by hand, which is doable but very vulnerable to human error.

Locally with a version control tool As with the previous point, locally handled version control only works if the project is handled completely by a single person (and backups are taken regularly). The use of a tool frees up the developer's time and they can focus more on the development when the versioning is handled automatically.

Centralized version control Centralized version control allows multiple developers to work on the same project.

Distributed version control This also allows multiple developers to work on the same project. Additionally it allows for different workflow setups than centralized version control systems.

4.3.6 Test Management

The customer affects greatly how tests are managed within Atostek Oy. If a customer requires a test document to be handed over at the end of the project or sets any other kind of demands for the testing process, deciding on a test management system and testin procedures is easy. If a customer says they have a separate testing team, formal testing might not be conducted at Atostek Oy at all. In these cases features are tested in an ad hoc manner without any planning. Some projects set up a test environment or tool despite the customer not specifically requesting this.

Test management and tests can be handled in many different ways:

Test document Test documents are mentioned in ISO 9000-3 standard [17] and therefore they are a valid and often required piece of documentation. The tests described in the document can be executed manually or automatically.

Unit tests / Test Driven Development In test driven development unit tests are written before the actual implementation of the code, which makes the

testing phase seem shorter while making the development time seem longer in comparison. Unit tests are often executed automatically.

Build server A build server can automatically execute unit tests and scripted tests on a pre defined schedule. The setup can take a while but it usually makes testing much easier.

Test tools There are also tools that help with test management (and often defect management on the side). These can handle manual tests, automated unit tests, and automated graphical user interface tests though it is not guaranteed that they handle all of these. TestLink and IBM Rational Quality Manager are examples of test management tools.

4.3.7 Defect Management

Despite being considered important, defect management is not handled as a separate concept in Atostek Oy. It is practically a part of task management since a task for fixing each defect is created as soon as a defect surfaces. Since each task is to be tested before marking it as complete, no special procedures are required for defects.

Defect management can be handled in much the same way as requirements and task managements.

A defect document Like the tasks, defects are usually short lived and it is unknown how many defects there will be. Therefore *a document* is more likely to be sensible to keep up to date than *a chart* or *a poster*. However even a document can be tedious to keep current.

A tool Defect management can be handled either with the same tools as requirements and task managements – tools such as JIRA, Mantis and Redmine. Because defects are essentially tasks, it is sensible to use the same tool for both. If a tool is used for test management, defect management can be handled with that as well.

Notes, email, and memory In a very small project it might be possible to manage defects in one's own notes, email, or memory. This is not recommended.

4.3.8 Risk Management

Risk management consists of two parts: figuring out what the risks are and preparing for the possibility of the risk becoming reality. At Atostek Oy it is not considered to be a separate aspect of project management although risk management is taken into consideration in all projects – more passively in some than others. Using *a checklist*

can help project managers in this. Table 2.1 on page 14 shows a checklist based on Boehm [18]. Project managers should not just rely on it alone but to add to it risks they think apply to their projects.

5. CASE STUDIES

As stated in the previous chapter, Subversion is the de facto version control tool in Atostek Oy. Three multivendor projects in Atostek Oy are in the process of migrating from Subversion to a distributed version control system. Each project is slightly different from the others and requires paying attention to different details.

For each case the baseline – the point where the project is before the change of version control mechanism is investigated – is described. Then the reasons for change are explained. Different solution options are examined and the chosen solution as well as the steps to migrate to that solution are laid out.

Atostek Oy is the only party present in each case. The customer and other vendors – referred to as "the Customer" and "the Vendor" in all cases – are different.

5.1 Case Study 1 - From Multiple Version Control Systems to One

In Case 1 the project has three parties: the Customer, the Vendor, and Atostek Oy. The Vendor develops hardware and corresponding software. Atostek Oy in turn develops different kinds of software that allows the end user to, via a graphical user interface (GUI), use the Vendor's hardware in combination with other hardware. The vendors communicate with each other through the Customer.

5.1.1 Baseline

Currently both vendors have their own internal version control system. For Atostek Oy this system is Subversion. When a release is made they deliver the release to the Customer and add the source code to the Customer's Source Gear Vault (SGV) system.

Tasks are managed with a task management software that the Customer hosts on its servers. Both vendors have limited access to these servers – they can access their projects on the task management system but nothing else. This way the vendors can manage the tasks freely adding and changing tasks when needed and the Customer can add tasks when they find bugs. This ensures that the Customer can have up to date information on what the vendors are working on. Most importantly this helps with communication between the vendors and the Customer: if more information is

needed on a specific task, the discussion can be had in the comments of that task. In that way it is easily accessible to everybody interested in that task.

However, because the task management is on the Customer's intranet and the version control systems are on the vendors' respective intranets, it is not possible to connect the task management system to the version control system so that the task management system could automatically figure out the version of the software that implements a task. This would help development and especially maintenance. If a bug is found relating to a certain task the traceability of the task would make it easy for the developers to decide where to start looking for the cause of the bug.

The version control system in the baseline is depicted in Figure 5.1. For simplicity's sake the task management software is left out of the picture.

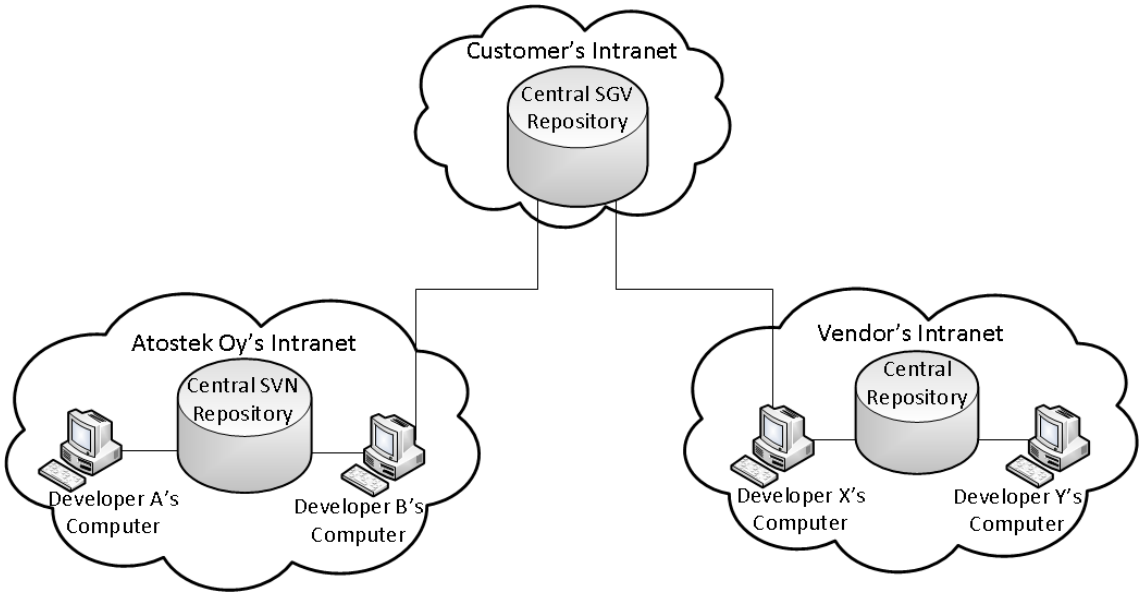

Figure 5.1: The baseline of Case 1. Atostek Oy uses Subversion (SVN) and the Customer Source Gear Vault (SGV) for version control. The Vendor's version control system is unknown.

5.1.2 Reasons for change

The main reason to change the current system is the opposition of Source Gear Vault. Since Source Gear Vault is in the Customer's intranet, vendors need to have internet connection and a configured access to the Source Gear Vault server. There was an incident when the Customer changed some of the settings on their end and it took two days and numerous phone calls to the Customer's IT support to get Atostek Oy's connection back up again.

Source Gear Vault is also very difficult to use when compared to other version control systems. This is highlighted by the fact that the vendors only use it rarely, which means they have to essentially relearn to use the system each time they use it.

This often causes mistakes to be made – mistakes which could be avoided if a more familiar and easy to use system was used. Also copying code from a vendor's version control system to Source Gear Vault is often seen as part of the release making process. However the process takes long enough to justify it being its own task.

Since code is added to Source Gear Vault only when a release is made, the comments associated for each version in Source Gear Vault are all along the lines of "Source code for release X.Y.Z". The Customer has to look elsewhere if they want to know what changes are included in each version. The vendors refer to their own version control systems so they do not have the same problem.

Since Atostek Oy's software uses the interface of the Vendor's hardware, it has a need to know what all has changed between releases. Sometime smaller changes are forgotten to add to documentation and if these cause bugs, it is difficult to figure out whether the bug is caused by the code done by Atostek Oy or the Vendor. The way Atostek Oy interfaces or plans to interface with the Vendor's hardware might impose requirements for the Vendor. Hence it is important to know which requirements are met in each release.

5.1.3 Different Solution Options

The fact that the Customer wants to have the source code as well as the releases of the final product affects the solution options greatly. Any change to be made has to make the transfer of the source code from a vendor to the Customer easier than it currently is.

A: Mirrored Repositories

The first and easiest option in the vendors' point of view, is to keep using the same version control systems they are using now and to mirror their repositories to the Customer's server. This way the developers need to learn to use only one version control system. Since the repository is mirrored, the developers need not use time to transfer the code to the Customer. This is shown in Figure 5.2.

If the connection between a vendor and the Customer's intranet breaks, it is easier to re-establish a connection due to the system being more familiar to all parties than the current one. At least Subversion, which is used by Atostek Oy, is more widely used than Source Gear Vault which means it is easier to find trouble shooting tips online if needed.

The entire version history with each commit is accessible to the Customer. If the versions, of which a release is made, are tagged, the Customer can easily see which changes have been done for each release from the version history alone. In Atostek Oy the tagging of released versions is already a common practice. The Vendor might

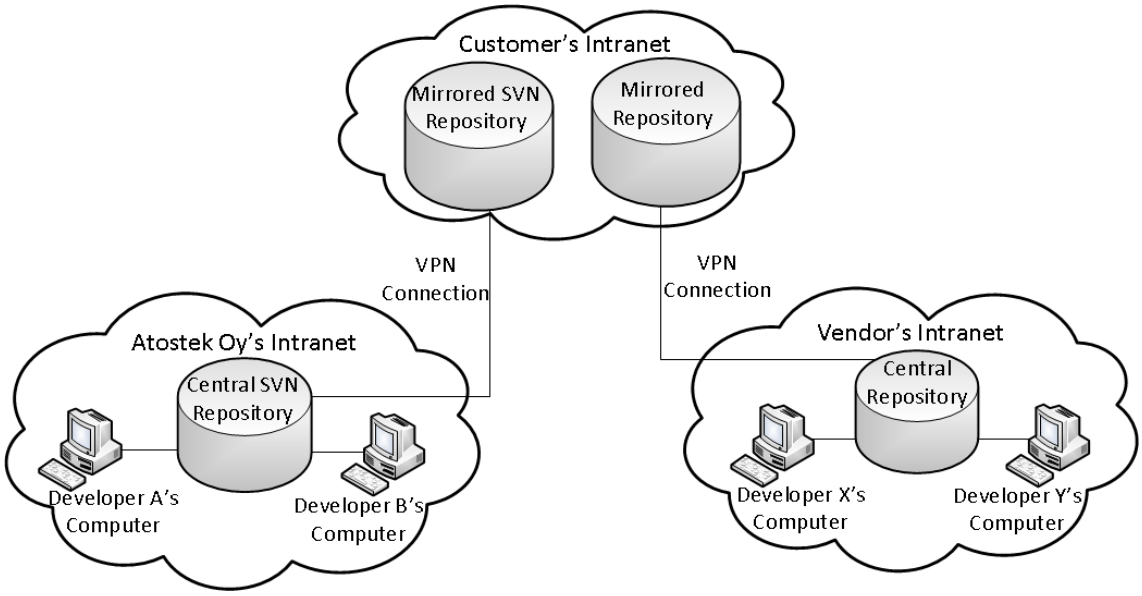

Figure 5.2: Case 1 with mirrored repositories. Atostek Oy uses Subversion (SVN) for version controls. The Vendor's version control system is unknown.

or might not do this now. If not, they would have to adapt this practice.

The downsides are mostly on the Customer's end. As opposed to only the released versions of code, all versions would be delivered to the Customer. Finding the released versions would take more effort than it currently does. Assumedly they most often want access to the latest released version of code, when they want to access source code at all, so if there are any changes made after the latest release, the Customer has to look for the latest released version.

Also it is not known what version control system the Vendor currently uses. If they do not use Subversion, the Customer would have to learn to use two new version control systems.

B: Everyone Uses a Distributed Version Control System

Distributed version control systems makes it possible to use multiple repositories, so both vendors and the Customer can have their own master repositories. Because the repositories need not be mirrored, the vendors can develop software in the same manner as always, but the transfer of source code to the Customer is much easier. The Customer's repository would only have the source code of the released versions and not the development versions in between releases. This solution is depicted in Figure 5.3.

The development of the software would essentially stay the same: there would be a repository in each vendor's intranet that acts as the master repository for development. Developers would pull the latest version from that repository and push their changes to it. As a bonus distributed version control systems make branching

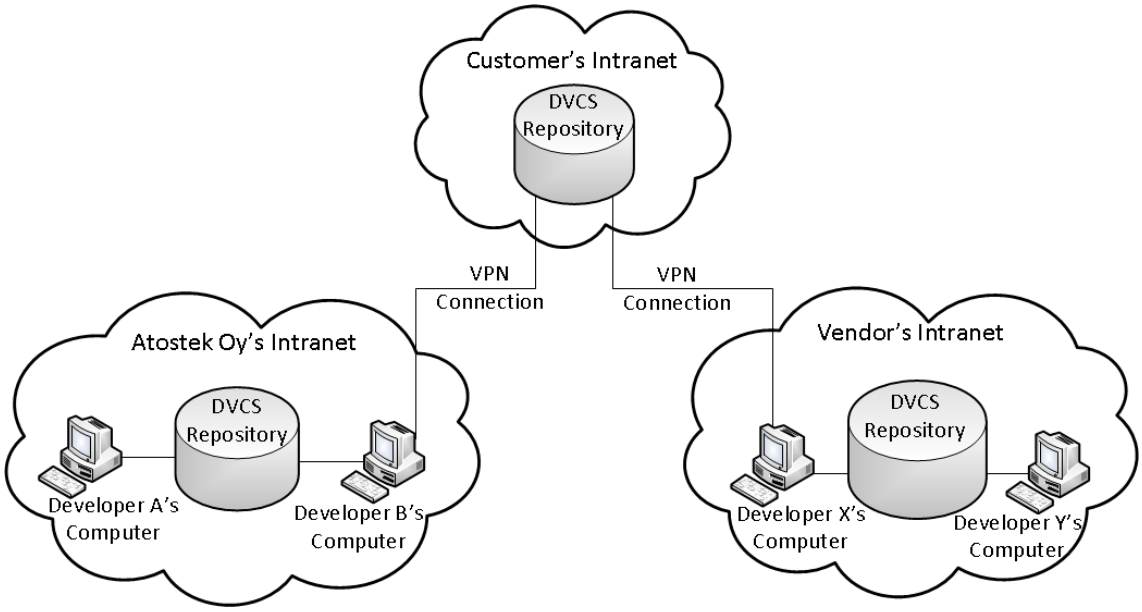

Figure 5.3: Case 1 with everyone using a distributed version control system (DVCS).

really easy, which would help in development – each task can be implemented in its own branch. For smaller tasks this can be done in a single developer's computer, for larger ones a branch can be added to the master repository where multiple developers have access to it.

When a release is made, the released version is tagged and pushed also to the Customer's repository. It is possible to add the releases of the software in the repository too so the handover process would be greatly simplified.

As an added perk distributed version control systems copy the entire repository to each computer. This means that even if there are connection problems between servers, there is no down time in development even if handovers cannot be done immediately. This is in line with Atostek Oy's policies of not needing internet connection in order to work.

5.1.4 Chosen Solution

Out of the two proposed solutions, the latter is easiest for the Customer (as opposed to being easier for the vendors). Despite forcing the vendors to learn to use a new version control system, it also makes development of the software easier so the vendors benefit from the solution as well. Git was chosen as the specific distributed version control system to be used.

The Customer is still hesitant so it has been agreed that the migration to the new system will be done in four stages:

1. Atostek Oy migrates their development to Git.

2. The Customer creates a repository that Atostek Oy can push their source code and releases into.

3. The Vendor migrates their development to Git.

4. The Vendor starts pushing their source code and releases into the same repository on the Customer's server as Atostek Oy pushes theirs.

Currently the migration is in stage one. There are many projects that Atostek Oy has developed for the Customer and some are worked on more often than others. Most of the projects that are currently active have been migrated from Subversion to Git. The goal is to have all projects migrated, but currently there has not been enough time between development tasks to do this.

Everyday practices to using Git are still being developed in Atostek Oy. When migration to Git started, everyone started using Git in the way they felt was best – no official practices were in place. The developers can use Git on their own computers ever which way they want but in order to have fruitful collaboration, agreed upon practices are needed.

The plan for official practices is derived from Gitflow [6]. In Gitflow the "Master" branch has only the official released versions of the software. There is a "development" branch, or branches, for development, "feature" branches for larger features, a "release" branch for the making of releases – including but not limited to testing and small final fixes. There is also a "hotfix" branch for maintenance tasks.

Figure 5.4 depicts the proposed workflow. The main difference between our proposed workflow and Gitflow is that the Master branch (coloured gray in Figure 5.4) is not added to – its head always points at the root of the version graph. The use of the Master branch in the same way as in Gitflow is still under consideration – we are still in the process of trying to determine the pros and cons of using the Master branch the proposed way and the Gitflow way. Other than that the proposed work flow would follow the principles of Gitflow: Each version is developed in its own branch (coloured orange in Figure 5.4), releases are created in their own branches (coloured green in Figure 5.4), and features in their own branches (coloured blue in Figure 5.4).

5.2 Case Study 2 - Expanding Version Control to Cover Design Documentation as Well as Source Code

Like Case 1, Case 2 has three parties, the Customer, the Vendor, and Atostek Oy. Atostek Oy and the Vendor work on different parts of the same software. The Customer creates configuration files for the software to use.

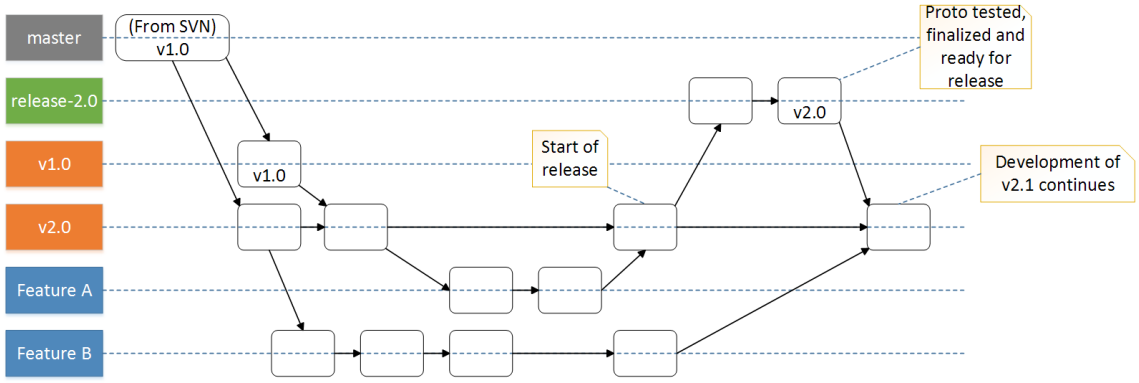

Figure 5.4: The proposed workflow for Git.

5.2.1 Baseline

At the moment each vendor has their own version control systems that are not connected to the others' systems in any way. For Atostek Oy this system is Subversion. The software is designed so that the vendors do not have to access each others' source code. The changes needed in the interfaces are discussed with the representative of the Customer who then relays the information to the other vendor. When a new version is to be released the Vendor delivers their binary files to Atostek Oy who creates the release, which is then delivered to the Customer. The Customer delivers new configuration files to the vendors as they are created so the software can be tested with them.

The development team at Atostek Oy is currently small – only one or sometimes two people work on the project. In the past the team has been larger, but currently the project consists mostly of maintenance. It is possible that the project grows in the future and more developers are needed again.

For a long time Atostek Oy and the Vendor had no direct contact with each other and all communication was done through the representative of the Customer. Now however, Atostek Oy works at times on the Vendor's code due vacations and deadlines. In these cases the needed source code files are emailed to Atostek Oy who emails them back after making the changes needed.

Atostek Oy has integrated their task management system to their version control system. This makes it easy to track in which version a certain task has been implemented. The task management system is only used by Atostek Oy. If the Customer adds new requirements or reports bugs, these are added to the task management system by the developers at Atostek Oy.

The Customer wants everything to be documented. When a new feature or idea is introduced a document is created and used as the basis of discussion. It is updated as new decisions are made. This is very useful because the documents are used later by developers. The same applies to manuals when the graphical user interface is

discussed.

However, the fact that the documents are version controlled manually is a problem. Each file is created so that it is opened in read-only mode. The document's name consists of a descriptive name for the document, the date when the document was created or changed in YYMMDD format, and the version number for that day all separated by an underscore. So for example: GUI-plan_130830_3.docx would mean the document is a plan for a graphical user interface and this version is the third one created on the 30th of August 2013. The documents are archived on a server which is regularly backed up. The directory structure is created by the project manager based on the logic he sees best. The transfer of the documents between a vendor and the Customer happens via email with the documents as attachments. The two vendors rarely work on the same documents but when they do, they go through the representative of the Customer.

The baseline of Case 2 is shown in Figure 5.5. It shows only Atostek Oy and the Customer's intranets as well as connections between them. The situation is mirrored with the Vendor.

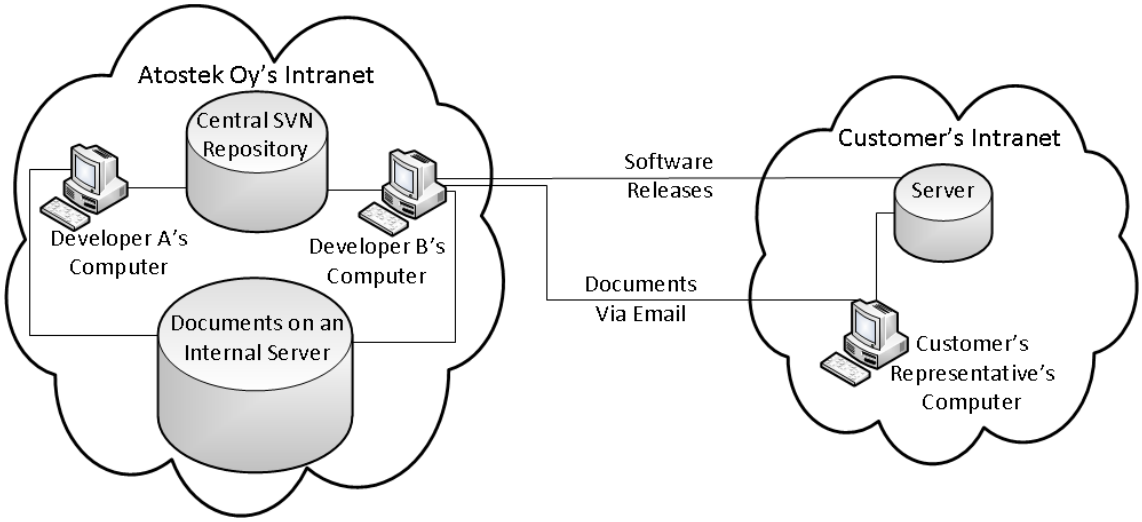

Figure 5.5: The baseline of Case 2. Atostek Oy uses Subversion (SVN) for version control of the software.

5.2.2 Reasons for Change

There are many reasons for changing the current version control mechanism. First and foremost keeping track of different versions of documents and the occasional source code file emailed over by the Vendor requires a lot of concentration. The documents are created in a way that makes it harder to overwrite an existing version of the document, but having them under a version control system would be even easier. Handling the version controlling of the documents by hand has resulted in

a monstrous directory structure with Archive directories full of past versions of documents that are rarely looked at.

The Vendor's source codes are only needed for a short time and usually the changes made are small so there will be only a few new versions. However if Atostek Oy had access to the repository the Vendor uses, accessing the code files needed would be easier and the version control system could keep track of each version created more easily than in the current situation. The Vendor has a system of keeping the version history of each file in comments at the beginning of each file. It seems to be created automatically by their version control system. Since Atostek Oy is not linked to that system, the developers there try to mimic the style and add their changes to the comments by hand.

Currently everybody needs to go through extra steps in order to create a release. The Customer has to track the changes to its configuration files and email them over to both vendors. The Vendor needs to track changes to their source code, create the binary files and deliver them to Atostek Oy. Atostek Oy needs to gather all necessary parts (the delivered configuration files and binary files as well as their own part of the software) and create a release, which is then delivered to the Customer.

Since the communication between vendors happen through the representative of the Customer who does not have the same technical background as the developers of each vendor, it is sometimes difficult to decipher what the message from the other vendor is. On the other hand, since the discussion always flows through the representative of the Customer, he knows exactly what is happening in the project and what they are getting for their money.

Since both vendors work on the same software, it is sometimes difficult to determine for whom it is easier to fix a bug. This has often resulted in lost time when one or both vendors go through their code and try to determine where the problem is. This has sometimes resulted in emailing source code files back and forth between the vendors.

5.2.3 Different Solution Options

There are many things to be considered when thinking about changing the way version control is done in Case 2. The most interest is paid to the version control of documents. However when the system might change anyway, it is sensible to look at the possibility of bettering the entire system.

A: Documents Are Added to Current Version Control Systems

The first and obvious idea is to add documents to the current version control system. This does not require big changes since everyone knows how to use their respective

version control systems. In addition this change happens purely inhouse so Atostek Oy can do this change without it affecting the practices used by the Customer and the Vendor. The solution is shown in Figure 5.6.

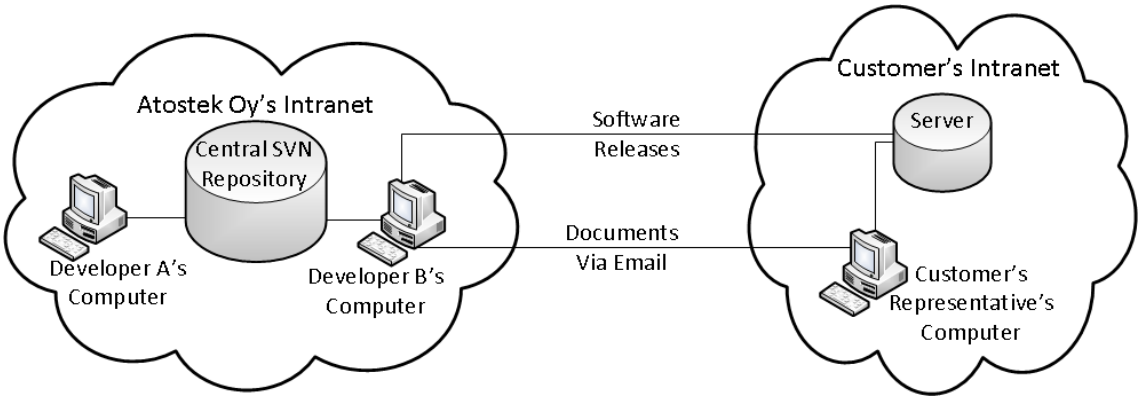

Figure 5.6: Case 2 with documents added to current version control systems. Atostek Oy uses Subversion (SVN) for version control.

The benefits of the solution are that since there is no need to learn how to use any new version control systems, adding the documents under a version control system is easy. The downsides are that the documents and source code files still need to be emailed back and forth. Also, this solution does not try to better the current version control system in any way.

B: Documents in Google Drive

Having the documents in Google Drive makes it possible for many people to edit the same file at the same time so that everyone sees the changes as others are making them. This makes it possible for example to conduct teleconferences during which they can all look at and edit a document they are working on. This is a considerable improvement to the current situation where only thing to do is to scribble notes during a teleconference – the client insists on talking over the phone instead of using newer teleconference systems – and the document updated based on these notes. Even when either the representative of the Customer has to visit a vendor's office or vice versa the document would still be updated based on the notes written down during the meeting. In both cases the updated document would then be emailed back and forth to check everyone understood each other. Google Drive even tracks changes made to the documents, which is a prerequisite to any solution. Figure 5.7 depicts this solution.

Despite having a lot of potential there are big problems with the use of Google Drive. First and foremost it is not secure – the documents are confidential and Google Drive is an external service provider. There is no non disclosure agreements between Google and its users. Also the change tracking is not completely reliable.

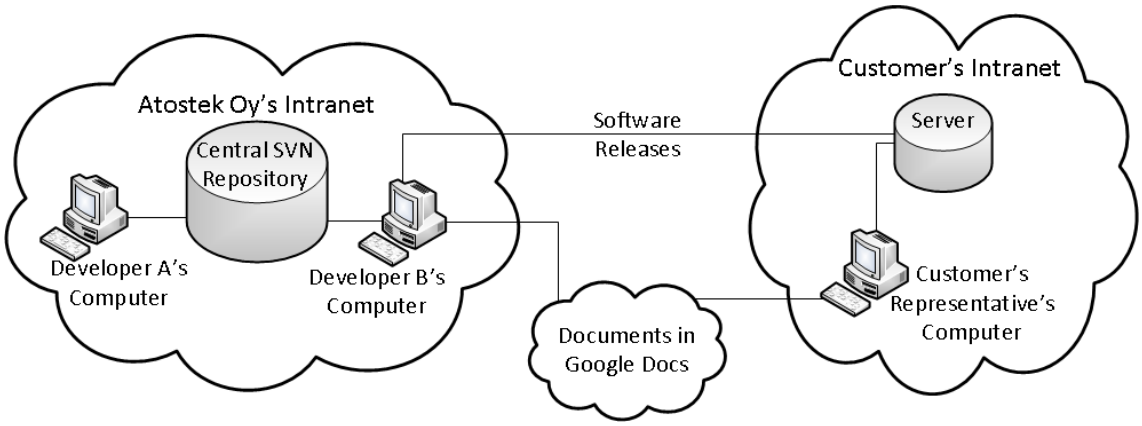

Figure 5.7: Case 2 with documents in Google Drive. Atostek Oy uses Subversion (SVN) for version control of the software.

Initially changes are tracked every few minutes and later grouped together. If space is running out the oldest versions are destroyed to free space for new versions. Using Google Drive also means that internet connection is needed more often than with the current system or other version control systems. This solution does not affect the version controlling of the source code in any way.

C: A Vendor and the Customer Share Their Documents in Common Version Control System

Having the documents in a version control system used by the vendor and Customer makes it much easier to handle the version control and delivery of the documents. In the past the vendors have been isolated from each other so it is reasonable to consider keeping them isolated now as well. The idea is to set up two repositories; one for each vendor. The Customer would have access to both repositories. This way the documents are in one place, version controlled and the latest version is accessible to everyone at all times. The setup is easiest to create with a distributed version control system such as Git or Mercurial. Using a potential internet connection problems are minimized. The solution is shown in Figure 5.8.

This solution removes the need to email documents back and forth – everyone has access to the latest version of every document. However email notifications of new versions of a document will probably be sent instead. It also upholds the tradition of keeping the vendors separate so the vendors make decisions the Customer is unaware of.

However everybody needs to learn to use at least one new version control system and the vendors need to use two version control systems synchronously. In a worst case scenario the Customer needs to learn to use three different version control systems. This solution also does not consider source code and the need to email it

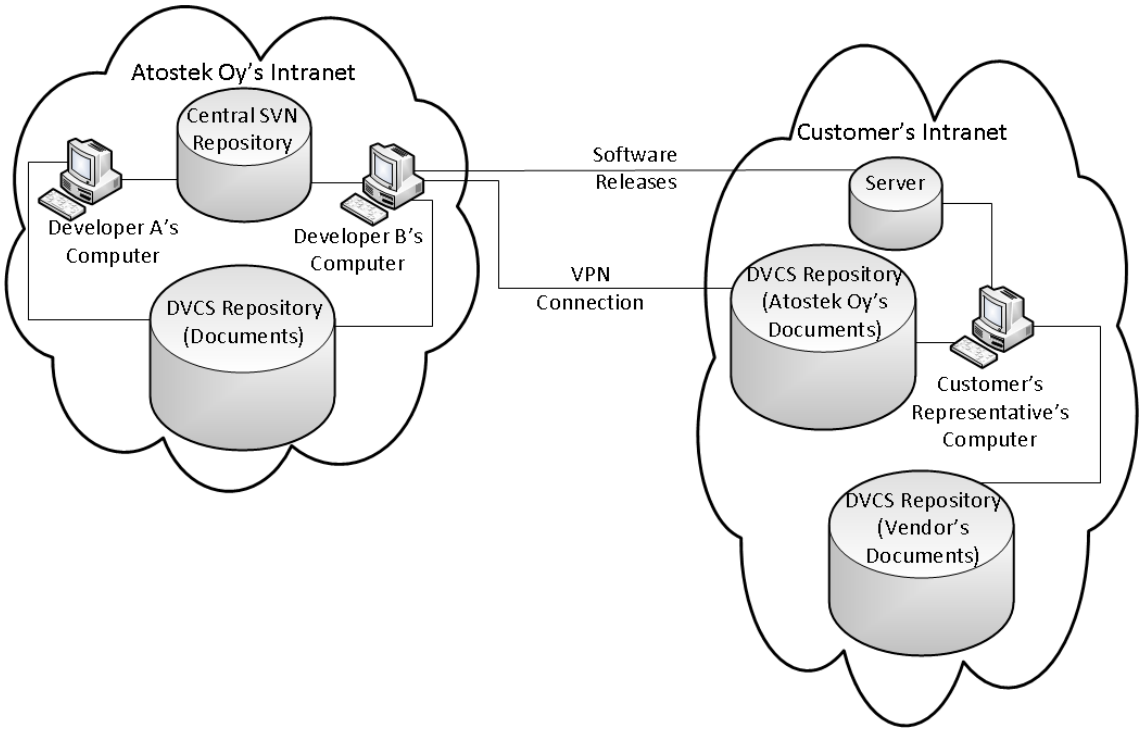

Figure 5.8: Case 2 with documents in a shared distributed version control system (DVCS) repository. Documents are shared only between one vendor and the Customer. Atostek Oy uses Subversion (SVN) for version control of the software.

between vendors at all.

D: Documents in a Shared Version Control System

One solution is to have documents in a Git repository that is shared by all parties. This way both vendors can see what the other vendor is doing and how they are doing it and the Customer has all the documents in a single place. Everyone can also access the latest version of any document at any time. This would be easiest to do with a distributed version control system. Source code would be version controlled in the current way. Figure 5.9 shows the cnfiguration of this solution.

This solution is not very much different from the solution in which the documents associated with different vendors are in different repositories. The documents need not be emailed between a vendor and the Customer though email notificiations of new versions will be sent. A positive difference is the fact that both vendors see the specifications of the parts of the software the other vendor is working on. This has the potential to aid in the search of the source of bugs that appear in the interface of the different parts of the software.

However both vendors still need to use two different version control systems. This solution also does not consider source code and the need to email it between vendors at all.

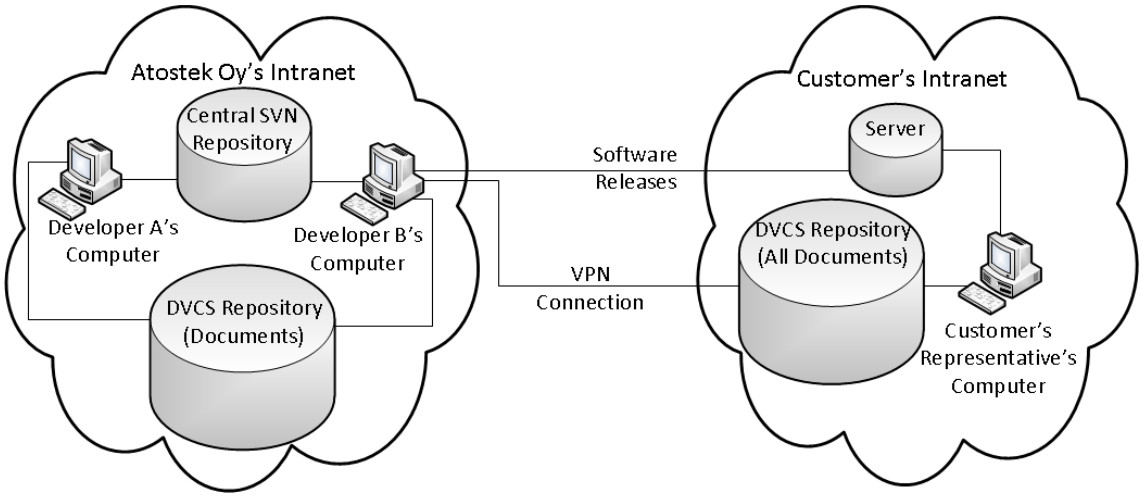

Figure 5.9: Case 2 with documents in a shared distributed version control system (DVCS) repository. Documents are shared only between all parties. Atostek Oy uses Subversion (SVN) for version control of the software.

E: Everything in a Shared Repository

A solution that addresses both problems – the need to email documents and source code files – is to have everything in a shared repository. Distributed version control systems allow this more easily than centralized version control systems. The solution is pictured Figure 5.10.

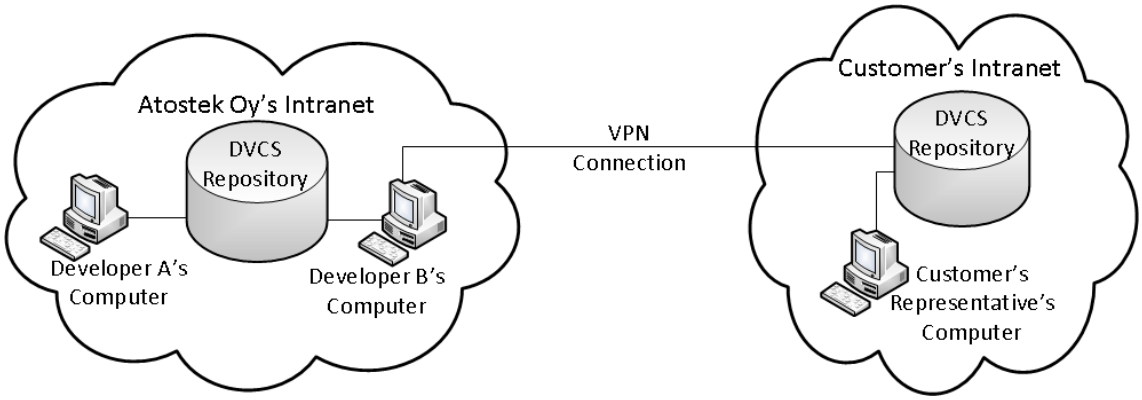

Figure 5.10: Case 2 with everything in a shared distributed version control system (DVCS) repository.

Like all solutions that involve using a distributed version control system, this requires new practices to be thought of and implemented. However this solution has all the same positive outcomes as the solutions that have documents in a shared version control system and it also eases greatly the occasions during which a vendor needs to work on the other vendor's source code. The code files need not be emailed between developers and the version history is automatically maintained. However as with the documents, emails containing attachments will most likely be replaced,

at least to some degree, by email notifications of new versions.

5.2.4 Chosen Solution

Having everything in a shared repository solves the problem of keeping track of the versions of the documents. It also makes it easier to work on the other vendor's source code when the need arises. Hence solution E is chosen.

A shared repository between all parties is easiest to implement with a distributed version control system. Since Atostek Oy currently uses Subversion (a centralized version control system) for version control, the chosen solution improves the processes of daily work, because branching is much easier in distributed version control systems than centralized version control systems. This is because, like in most large and long term software projects, there will be functionality that is better to implement in its own branch separate from the other, often more stable, versions of the software.

The schedule of the migration is still under discussion. The Customer uses Mercurial internally so the vendors will migrate to using it as well.

Once the schedule of the migration has been decided it will be executed in the following stages:

1. Atostek Oy migrates their documents to Mercurial.

2. Atostek Oy's document repository is copied to the Customer's server.

3. The Vendor migrates their documents to Mercurial.

4. Atostek Oy and the Vendor migrate their source code to Mercurial.

Although currently the team working on the Customer's project consists of only one person at Atostek Oy, it is important to decide on proper procedures for using Mercurial. If more developers are added to the project or the person in charge of the project changes it is vital to have procedures in place to avoid confusion. As of writing this the procedures have not been decided yet.

5.3 Case Study 3 - Migrating Due to Customer's Wishes

Case 3 is larger than cases 1 and 2 at least in the sense of vendors – there are five in total. Each vendor works on a separate software package, which are bundled together right before a release.

5.3.1 Baseline

Since each vendor works on a distinctly separate software package, it is unsurprising that each vendor has their own version control system that is separate from the

other vendors' version control systems. When a release is to be made, each vendor delivers an installation package of their software to the Customer. The Customer actually sends these packages to Atostek Oy where the installation package for the entire software bundle is created. The baseline is pictured in Figure 5.11.

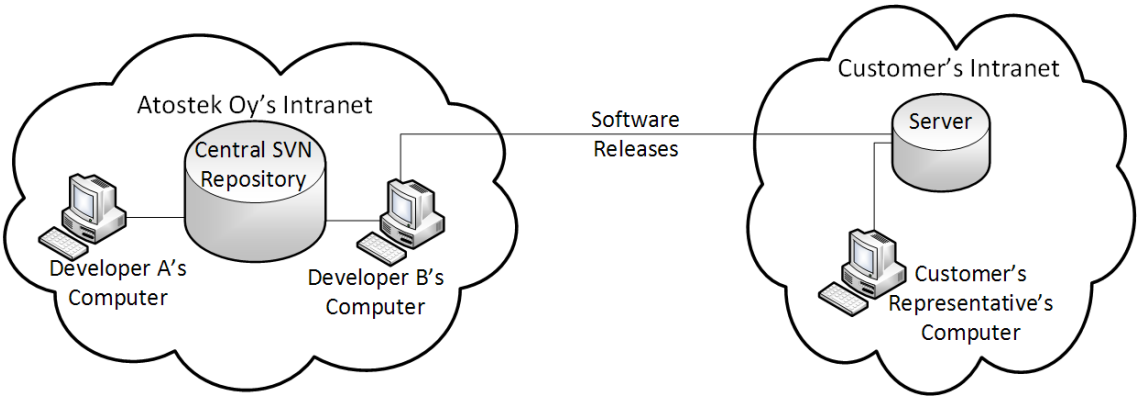

Figure 5.11: The baseline of Case 3. Atostek Oy uses Subversion (SVN) for version control.

There are actually two distinct products that are developed in parallel at Atostek Oy. The products are almost the same – the main difference is the branding. In order to better keep track of the development, a release of both products is made when one is released. That way the version numbers are always in sync.

Atostek Oy has integrated their task management software to their version control system (Subversion). This has proved an excellent practice and the project management has become reliant on this. Not only does the integration mean that it is easy to find out what task has been implemented in which version, but it also keeps track of what version of each software is included in each release.

5.3.2 Reasons for Change

The Customer wants Atostek Oy to start using Mercurial and use a shared repository with the Customer. Using a distributed version control system makes collaboration between Atostek Oy and the Customer easier. Mercurial was chosen since the Customer already uses it. The future setup is pictured in Figure 5.12.

5.3.3 Migration Issues

The Subversion repository currently used by Atostek Oy is so large everything cannot be moved to the Mercurial repository. Therefore all essential things – the branches that are still developed – will be moved after which the Subversion repository will be left in read-only mode.

Using a large number (over 2000) of named branches in a Mercurial repository impairs its performance [3]. Due to this the Customer has used cloned repositories

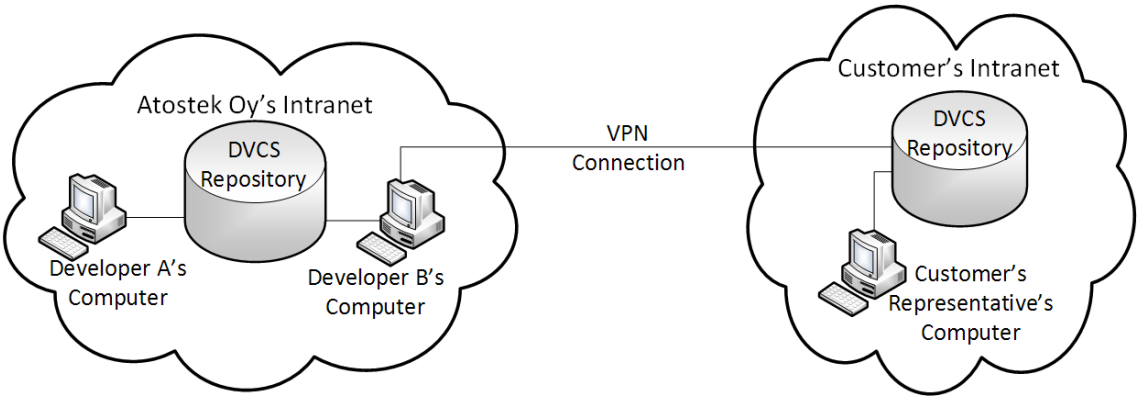

Figure 5.12: The solution of Case 3 where a distributed version control system (DVCS) is used.

instead of branches. Evidently the features they implement are small enough to always be handled by a single developer.

Atostek Oy, however, works on features which are too large for one developer to handle. The features are also developed over a long time. On top of this, Atostek Oy develops multiple versions of their software simultaneously. It is important to keep track of which feature or version of the software is worked on.

Therefore Atostek Oy suggests that each version of software will have only one named branch (ex. "10.1.2"). Features which have multiple people working on them should have named branches as well. Smaller tasks can be done using anonymous branches, bookmarks or cloned repositories – it is up to the developer how they handle smaller features and version control in general on their computer. This way each version and larger feature is easily identifiable and accessible, while the number of named branches is controlled.

Deciding when to create a new named branch – for example a new version – requires a bit of guess work. It is easier and safer to not make new branches if it is suspected that the branch would be short lived. If it turns out the branch is not short lived, a branch can be created later on. This is not the optimal way but it is not possible to always predict the future.

Branches need to be merged in a certain order to get all required branches to contain the changes. The branch is first merged to the second oldest branch (in relation to the branch being merged) which is then merged to the third oldest branch etc. until all required branches contain the merged information.

The Customer has used cloned repositories instead of branches. They are reluctant to allow Atostek Oy to use named branches partially because of the limit and partially because they fear their procedures would change drastically. The discussion to find a procedure that suits everybody is still ongoing.

6. EVALUATION

Different projects have different needs in version control as well as other aspects of project management. The three cases described in Chapter 5 depict three situations in which version control systems were chosen based on different criteria. Table 6.1 shows how each system compares against each criteria. The comparisons are also written in more detail below.

In most cases the different version control systems are discussed in general – for example distributed version control systems will be discussed as a group and not as individual systems such as Git or Mercurial. However there is only one system that allows simultaneous file editing with version control – Google Drive. It will be therefore impossible to discuss these systems in general, so instead Google Drive will be focused on specifically. Google Drive has not worked as a code repository, but code files are only one example of filetypes that need to be versioned, which is why Google Drive is discussed.

6.1 A Varying Number of Developers

In many software projects the developers working on the project at the beginning are not the same that work on it toward the end. Especially in longer projects some developers switch projects or even companies during a project's life cycle. Sometimes the amount of developers stays the same, sometimes their numbers wax and wane according to the scheduling demands of the project. The version control system used can support a varying amount of developers or it can impose restrictions.

Version control by hand (+) When handling version control by hand it is important to have clear rules, procedures and routines in place to ensure it is done properly. The more people work on the project, the more important it is to have clear procedures, to avoid simultaneous file editing for example. Generally the more developers working on the project, the more communication is needed.

Local only version control (-) Local only version control sets strict limits to the number of people working on a project – versioning is done on one computer only so only the person working on that computer has access to the version control system. This can be circumvented by using the computer in shifts or transferring code between computers on a memory stick, but more sophisticated version control systems offer a better alternative.

Table 6.1: Evaluation of different version control systems. Areas in which a system does well are coloured green, areas in which a system does not do especially well or poorly are coloured yellow, and areas in which a system does poorly are coloured red.

	Version Control by Hand	Local Only Version Control	Centralized Version Control	Distributed Version Control	Simultaneous File Editing With Version Control
A varying number of developers	+	-	+	++	+/-
Internet connection required	+/-	+	-	+	-
Access from multiple intranets	-	-	+/-	+	++
Support for different file types	+	+	+	+	-
Integration to other project management tools	-	-	+	+	-
Merging	-	+	-	+	+
Administration and security	+	+	+	-	+/-
Different workflow options	+	-	-	+	-
Ease of use	+/-	+/-	+	+/-	+
Need for communication	-	+	+	+	+
Performance	-	+	+/-	+	+
Backups	-	-	-	+	+/-

Centralized version control (+) When the repository is on a server many computers can access it and work on the same project at the same time. The fact that multiple people are interacting with the server possibly at the same time, does set certain demands on the hardware of the server but this usually becomes a problem only in very large projects.

Distributed version control (++) Distributed version control systems are designed so that users do not interact with the server unless they are to push or pull data. All other operations are performed on the developer's own computer. This already diminishes the demands for the server. With distributed version control systems it is also possible to use a server farm instead of a single server as the main repository, which allows much larger projects than a centralized version control system.

Simultaneous file editing with version control (+/-) Google Drive repositories are on Google's server farms so there is no way to affect the hardware by the users. However Google actively strives to provide the best user experience and as a part of that they make sure their servers can handle the traffic. However simultaneous file editing can be confusing and difficult if many people edit the same document at the same time and in the same part of the document without prior communication.

6.2 Internet Connection Required

In today's world internet connection is assumed to be available almost anywhere. However sometimes internet connections are down for various reasons. Depending on the company and the project this can be acceptable or unacceptable.

Version control by hand (+/-) Depending on how this is done, it can be easier or harder to continue working without internet access. If version control is done locally, there is no problem. If it is in the company's intranet, there is a chance that the necessary servers can be accessed even if there is no access to the internet. If the server cannot be accessed, people can still work on the files they have on their own computers. However depending on procedures and the length of time before people have access to the server, merging the changed files can become very difficult.

Local only version control (+) Since in local version control systems the repository is on the same computer the developer works on, having internet connection does not affect the connection to the repository in any way.

Centralized version control (-) If the developers cannot connect to the server that hosts the repository, they cannot check out or commit their work to the repository. They can however work on the files they have on their computer already. If the outage lasts a long time and multiple developers work on the same files, it could

be difficult to merge the files back together again.

Distributed version control (+) When a developer pulls from the master repository from the first time, they actually copy the entire repository to their computer. The repository is updated whenever the developer pulls the latest changes from the master repository. This means that even without a connection to the internet a developer has access to all the files and can therefore continue working normally. Because merging is made very easy in distributed version control systems, it is not difficult for the developers to merge all their changes back together once they can connect to the master repository again.

Simultaneous file editing with version control (-) Simultaneous file editing in Google Drive can only happen when the document is opened in a browser, so it is not possible to do without internet connection.

6.3 Access from Multiple Intranets

In the corporate world the master repository often resides in a secure intranet so only authorized people can access it. Sometimes it is important for a developer who is not currently in the intranet to access the repository. The most common way of providing this access is to set up a VPN connection between the server and the computer that requires access to the intranet. Once the VPN connection is set up, the developer can access all the files as if they were in the intranet.

Version control by hand (-) Once the VPN connection is set up, it is easy for the developer to access the files needed and continue working. However VPN connections are not always reliable and without it, it is impossible to access the server. A way to resolve this is to create a password protected web page which allows people to upload and download documents. It is less secure than having the files in a private network, but is secure enough to be used as an alternative to a VPN connection.

Local only version control (-) Since in local version control systems the repository is on the same computer the developer works on, there is no need or possibility to consider accessing it from different intranets.

Centralized version control (+/-) Once the VPN connection is set up, it is easy for the developer to access the files needed and continue working. However VPN connections are not always reliable and without one, it is impossible to access the repository. A way to resolve this is to put the repository in a public network behind a password. This requires more attention paid on the maintenance of the repository and it is less secure than having the repository in a private network, but is secure enough to be used as an alternative to a VPN connection.

Distributed version control (+) It is possible to set up repositories to each intranet that needs access to the master repository and sync them from time to

time. This way there is no need to have constant access between different intranets. Like with centralized version control the repository can also be put in a public network behind a password. In this case the benefits of that are not as great as with centralized version control systems.

Simultaneous file editing with version control (++) Since the files are always accessed via internet, it does not matter which intranet the file is accessed from.

6.4 Support for Different File Types

There is no limit to file types that need to be under version control in a given project. However some version control systems impose certain limitations themselves to the files that can be versioned.

Version control by hand (+) There are no limits to the type of files that can be version controlled by hand. Depending on the operating system or programs on the computer it might not be possible to open all files under the version control system by every developer's computer, but this does not affect the versioning of the files.

Local only version control (+) There are no limits to the type of files that can be version controlled.

Centralized version control (+) There are no limits to the type of files that can be version controlled.

Distributed version control (+) There are no limits to the type of files that can be version controlled.

Simultaneous file editing with version control (-) Google Drive has been found not to work as a code repository so it limits the possible files to be stored a great deal. However for files more associated with administration etc. – such as written documents, spreadsheets, pictures, surveys, and slide shows – are easily stored by it.

6.5 Integration to Other Project Management Tools

Integrating the version control system to other project management tools, such as task and defect management tools, can be very useful. However it is not possible to integrate all version control systems to these tools.

Version control by hand (-) When versioning is done by hand it is impossible to get it automatically paired with a tool. Any information shared with the tool needs to be put in by hand.

Local only version control (-) Tools created to help with other aspects of project management are so new they have focused on integrating with newer version

control systems.

Centralized version control (+) Many task management tools have add-ons that allow the most popular centralized version control tools to be integrated with them.

Distributed version control (+) Many task management tools have add-ons that allow the most popular distributed version control tools to be integrated with them.

Simultaneous file editing with version control (-) Since Google Drive is not designed to be used as a repository, it is not possible to integrate it with other project management tools.

6.6 Merging

Sometimes multiple people work on the same file at the same time. These changes are often all valid and needed in the final product, so all changes need to be in the final version of the file. This requires the changes to be merged together.

In a larger scale a new branch can be created so that a larger feature can be developed without risking breaking the official version of the program. When the feature is done, the branch will be merged back to the official version.

Version control by hand (-) Merging needs to be done by hand, just like everything else. There are tools to show the differences between two files, but this can still be very tedious.

Local only version control (+) Since only one person can use the computer at a time, there are no situation in which merging is required.

Centralized version control (-) The basic idea of centralized version control systems is that the central repository has the official version of the files. If someone tries to commit a version of the file that conflicts with the one in the repository, the system will see the new version as a threat. The developer who is trying to commit the conflicting file needs to resolve the conflicts, often by hand, before they can commit the new version of the file. This can be tedious with larger files with many changes and nightmarish when merging branches.

Distributed version control (+) Distributed version control systems have multiple repositories and no inherent way of knowing if one of them is more official than another. Therefore merging is seen as a natural and important part of communication between the repositories. There are mechanisms that make it easy, although everything cannot be handled automatically and there are times when the developer needs to solve conflicts by hand. However this principle makes merging single files and branches much easier than for example centralized version control systems.

Simultaneous file editing with version control (+) Since everyone can see the changes done to a file as the changes are made, merging is not an issue. However

if two people want to edit the same part of a document at the same time, editing might be difficult.

6.7 Administration and security

In some projects there is need to control who has access to which parts of the source code. Administrators can control for example security, access control, permissions, management of user accounts, etc. Some version control systems make the administration easier than others.

Version control by hand (+) When version control is done by hand, the administrators have full control over the security of the files.

Local only version control (+) The repository is on the same computer from which it is accessed. Therefore administration can be done by administrating the users of the computer. Security is handled once the security of the computer is handled accordingly.

Centralized version control(+) centralized version control systems have a central repository on a single server on which it is easy to do all administration and security setups on.

Distributed version control (-) Since the server or computer holding the repository is at the company's site, or on a trusted partner's site, the administrators have full control over the security of the servers. However, the most popular distributed version control systems – Git and Mercurial – allow the user to identify with any string. This can and often is remedied by policies stating that only valid email addresses etc. are allowed for identification purposes.

Simultaneous file editing with version control (+/-) Access can be set in three levels: only specified people listed have access to the file, only people who have the direct link have access to the file, or anybody on the internet has access to the file. The security of the files is questionable however. Google hosts the servers on which all documents are stored. Therefore the company utilising Google Drive has no say over the security.

6.8 Different Workflow Options

The workflows used in a project can vary greatly. All developers can be seen as equal and therefore all can have the same access to a repository. Or the project manager may need to approve all changes before they can be put to the official repository. Or some other workflow is desired.

Version control by hand (+) The workflow can be decided to be anything. Everybody may have access to all the files and make new versions of them freely. Or new versions of files can be sent to a person who reviews changes before adding

new versions for the others to see.

Local only version control (-) Everybody who has access to the computer and therefore to the repository are on the same level – if the access rights for the users are the same.

Centralized version control (-) Anybody who needs to elaborate on the project needs to have access to the central repository. And anybody who has access to the central repository has the same rights to checkout from it and commit to it.

Distributed version control (+) Since distributed version control systems support multiple repositories, there are many workflows to choose from.

Simultaneous file editing with version control (-) Anybody who has access to a file have the same rights to it. Since anybody can see the changes as they are made, it is impossible to have someone review changes before they are accepted.

6.9 Ease of Use

Ease of use is extremely important. While sometimes some feature is so important it is worth learning to use a new system in order to be able to use the feature, often the system which is easiest for everyone to use is chosen.

Version control by hand (+/-) The process and workflow for manual version control needs to be planned beforehand and it requires a lot of concentration before the procedures become habitual. However most people who feel the need to use version control know how to create, copy and rename files and directories so this method does not require the users to learn to use new tools.

Local only version control (+/-) Local only version control systems have not been developed since more complex centralized version control systems came along. Therefore they remain relatively simple, but they do not have graphical user interfaces.

Centralized version control (+) The idea behind a centralized version control system is fairly simple to understand. They also often come with a GUI which makes their use much easier for developers who are not comfortable working on the command line.

Distributed version control (+/-) The learning curve for distributed version control systems is generally considered to be much steeper than it is for centralized version control systems. This is understandable since distributed version control systems are more flexible in many ways and most people who climb the curve say it is well worth the effort. But for the uninitiated there is another problem: the graphical user interfaces are not as well tested and honed as they are for centralized version control systems since they have existed for a much longer time. There are GUIs – such as TortoiseGit and TortoiseHg – for those who prefer to use them. However many developers say they would rather use the command line option, because at

the moment even the best GUIs are too confusing.

Simultaneous file editing with version control (+) Google Drive takes care of saving and versioning automatically, so in the simplest case the user just edits the document and lets the versioning and even saving happen on the background. Everything, including reviewing the change history and recalling a past version is done via the simple graphical user interface provided.

6.10 Need for Communication

In any project where there are more than one people involved, communication is required. The version control system used can either work as a way of communication or create a need for more communication.

Version control by hand (-) Initially all procedures and workflows need to be explained to everyone involved. Then, depending on the procedures, it can be important to know whether someone is editing a file or not before opening it for editing. If a new version of a file is created or a new file added, other developers might not notice without informing them of it specifically.

Local only version control (+) If a person works on the project alone, there is no need for communication. If multiple people work on the same computer at different times, local only version control systems do help with communication. Checking out the latest version shows what has changed and the log messages, if written properly, describe the changes made. A developer can lock a file if it is important that nobody else can edit it before they have done all their changes to it. This might prompt questions on how long the file will be locked, but since this is assumed to happen fairly rarely, it does not significantly increase the need for communication.

Centralized version control (+) Like with local only version control checkouts show what has been changed and log messages help explain the changes made to each version of the software. In some centralized version control systems it is also possible to lock files if need be.

Distributed version control (+) Like with local only version control and centralized version control systems a developer sees what has changed when they pull latest changes from another repository. Log messages are important in these systems too. It is possible to send pull requests to other developers. That means that when an important change is made, other developers can be notified of this so they know to pull the changes to their repositories.

Simultaneous file editing with version control (+) Everybody can see the changes as they are made and version history is easy to see. However there are no log messages associated to each change, so the only way to see what has been changed is to actually go through the file and see the changes (they are highlighted).

Interestingly, because everybody can see the changes as they are made, this can be used as means of communication – a document can be used as a real time chatroom for the team.

6.11 Performance

Nowadays people are used to things working fast. Version control systems are a project management tool. Their point is to help with project management, not hinder it by taking up too much time.

Version control by hand (-) Each file needs to be version controlled by hand. Depending on the procedures in place this can be slow or very slow.

Local only version control (+) All operations are done locally on the same computer. This makes the use of local only version control very fast.

Centralized version control (+/-) Every operation is done against a server. This does not mean using a centralized version control system is tediously slow, but it does mean the operations will always take time.

Distributed version control (+) Depending on the size of the repository, copying the repository for the first time can take a long while. However once the repository is copied to the computer, most operations are done locally, which makes them really fast. Pushing and pulling changes to and from another repository is still done against a server.

Simultaneous file editing with version control (+) With Google Drive every keystroke is transmitted over the internet. But since the concept of simultaneous file editing over the internet is so novel, people are more tolerant with the slowness caused by this.

6.12 Backups

Version control is a great way of keeping track of changes and accessing earlier versions of software. However if the repository is lost, the information it held can naturally not be accessed. Some version control systems have inherent mechanisms that help with backups.

Version control by hand (-) Depending on the protocol versions that are worked on might be saved on a local machine and copied to the server where everybody has access to them. In that case some versions of some files are possibly saved if the server is corrupted. To ensure that no files are lost, backups of the server need to be made regularly.

Local only version control (-) The repository and local copies of files are on the same computer. If the repository is corrupted, everything is most likely lost. Making backups regularly is necessary.

Centralized version control (-) If the repository is lost, there are local copies of files on developers' computers. However the version history is only held on the server, so if there are no backups of the server, that information will be lost if the server is corrupted.

Distributed version control (+) In distributed version control systems the entire repository is copied to each computer that pulls from another repository. So even if a server containing a repository is corrupted, there are still repositories that have the entire version history stored on them. This does not mean, however, that making backups of the server(s) should be ignored – the repositories on developers' computers will be slightly different due to the developers working on different parts of the project. Additionally not all developers need all branches, so batching the lost repository together from different repositories could be difficult, or in a worst case scenario some branches could be lost completely. It is also not smart to rely on only one or some computers breaking at the same time.

Simultaneous file editing with version control (+/-) The files are hosted on server farms owned by Google. Google's business is based on providing services such as these so they take care to make backups of all data. However since Google Drive is not very secure, one should not rely solely on their backups [21].

7. CONCLUSIONS

Having multiple companies work on a single project is a common practice today. A successful multivendor project requires a lot of attention, planning and communication from the customer and the vendors. This can be seen from the way having multiple vendors affects each aspect of project management.

Atostek Oy takes part in many multivendor projects. There are many ways in which to manage these projects. The attitude and experience of both the project managers' as well as customers' affect greatly how the projects are run. These include conscious decisions such as which tools to use for certain tasks as well as sometimes subconscious decisions such as what aspects of project management to focus on.

One important decision is the selection of version control mechanism. A good version control mechanism can make the development of software much easier. Conversely selecting an unsuitable version control system can be a great burden on the developers.

The most popular version control systems today are either centralized version control systems, such as Subversion, or distributed version control systems, such as Git or Mercurial. In addition there exist local only version control systems, which are not very popular since most software projects involve multiple collaborating developers. There are also new types of version control systems being developed. An example is Google Drive, which allow multiple people to edit a file simultaneously while keeping track of changes made. It is also possible to not use a version control tool at all and do versioning by hand.

This thesis investigates how multivendor projects are managed in Atostek Oy. Specific attention is paid to three projects in which the version control mechanism used is questioned and changed in the middle of the project.

Each project is different. Therefore there cannot be a formula for the project management or even version control. The three cases studied in this thesis describe three different projects in which a version control mechanism is already set up and three different reasons why the mechanism must now be changed in order to make them better. Although in each project a distributed version control system is chosen, the reasoning behind the solutions is different. Thus it cannot be concluded that distributed version control systems should be used in all projects.

Each version control system has its pros and cons. Some version control systems

are increasingly outdated while new ones are created. This means that while the new systems might have features that the old ones lack, the old ones are more stable, established and familiar. Each system has its use and can be very powerful when used well. They can also be a nuisance if used in a situation where another system would perform better.

The different version control systems are examined and compared to each other in the light of points, ideas and requirements which arose in the case studies. This analysis is not perfect: The investigation could be more thorough. However this comparison can help in selecting the right system for different projects.

A checklist for project management is also introduced. It discusses briefly different ways in which each aspect of project management can be handled. It is not exhaustive, but can help both current and future project managements so see what project management comprises of and to decide how each aspect of project management will be handled in their project. In the future the checklist could be expanded and even changed to be a guide or a set of rules.

BIBLIOGRAPHY

[1] Chapter 11. multiple developers - several developers simultaneously attempting to run CVS. http://evscm.org/manual/html/Concurrency.html. [cited 31.5.2013].

[2] CVS - Concurrent Versions System. http://cvs.nongnu.org/. [cited 23.10.2013].

[3] Feature separation through named branches. http://mercurial.selenic.com/wiki/Workflows#Feature_separation_through_named_branches. [cited 26.9.2013].

[4] Git. http://git-scm.com/. [cited 23.10.2013].

[5] Git – about: Distributed. http://git-scm.com/about/distributed. [cited 11.6.2013].

[6] Git tutorials: Gitflow workflow. https://www.atlassian.com/git/workflows#!workflow-gitflow. [cited 21.8.2013].

[7] Gnu RCS. http://www.gnu.org/software/rcs/. [cited 23.10.2013].

[8] Google drive as source control? how can this be?! [cited 3.7.2013].

[9] Jira. http://www.atlassian.com/software/jira/overview. [cited 3.5.2013].

[10] Mercurial - work easier, work faster. http://mercurial.selenic.com/. [cited 23.10.2013].

[11] OS X man pages. https://developer.apple.com/library/mac/documentation/Darwin/Reference/ManPages/mann/rcs.n.html. [cited 23.10.2013].

[12] Rational quality manager. http://www-03.ibm.com/software/products/en/ratiqualmana. [cited 10.11.2013].

[13] Revision history. https://support.google.com/drive/answer/190843?hl=en. [cited 13.6.2013].

[14] Subversion. http://subversion.apache.org/. [cited 23.10.2013].

[15] Testlink open source test management. http://testlink.org/. [cited 10.11.2013].

[16] 610.12-1990 - IEEE Standard Glossary of Software Engineering Terminology. http://standards.ieee.org/findstds/standard/610.12-1990.html, 1990.

[17] ISO 9000-3, 2004.

[18] B. W. Boehm. *Software risk management*. IEEE Computer Society Press, 1989.

[19] T. Bonam. Mistake metamorphism. *Testing Experience*, (17):94, March 2012.

[20] K. Brennan. *A guide to the Business Analysis Body of Knowledge (BABOK guide), version 2.0*. International Institute of Business Analysis, 1. edition, 2009.

[21] A. Bridges-Smith. Do I need to back up Google Docs? Absolutely. http://spanning.com/do-i-need-to-back-up-google-docs-absolutely/. [cited 8.11.2013].

[22] S. Bucur, V. Ureche, C. Zamfir, and G. Candea. Parallel symbolic execution for automated real-world software testing. In *EuroSys '11 Proceedings of the sixth conference on Computer systems*, pages 183 – 198, New York, NY, USA, 2011. ACM, ACM.

[23] S. Chacon. Git pro - 1.1 getting started - about version control. http://git-scm.com/book/en/Getting-Started-About-Version-Control. [cited 11.6.2013].

[24] A. M. Davis. Just enough requirements management: Where marketing and development meet. http://www.sigs.de/download/oop_09/Davis%20Mo4%20brb%20WR%20A4.pdf. [cited 26.4.2013].

[25] C. Ebert. Techniques for successful software product management. http://www.executivebrief.com/software-development/techniques-successful-software-product-management/, January 2010. [cited 6.5.2013].

[26] S. Farooq and S. M. K. Quadri. An externally replicated experiment to evaluate software testing methods. In *EASE '13 Proceedings of the 17th International Conference on Evaluation and Assessment in Software Engineering*, pages 72 – 77, New York, NY, USA, 2013. ACM, ACM.

[27] I. Haikala and J. Märijärvi. *Ohjelmistotuotanto*. Talentum. 440 p, 11. edition, 2006.

[28] R. Hutchison and S. Shih. Pat. US 6651123B1. File system locking, 11 2003.

[29] IBM. IBM – Websphere Process Server, Version 6.1 – Life Cycle of Human Tasks. `http://publib.boulder.ibm.com/infocenter/dmndhelp/v6r1mx/index.jsp?topic=/com.ibm.websphere.bpc.610.doc/doc/bpc/ctasklifecycle.html`. [cited 26.4.2013].

[30] M. Jackson. *Requirements and Specifications: A Lexicon of Software Practice, Principles and Prejudices.* ACM Press, 1995.

[31] M. Kirousis, P. Spirakis, and P. Tsigas. Reading many variables in one atomic operation: solutions with linear or sublinear complexity. *IEEE Transactions on Parallel and Distributed Systems*, 5(7):688 – 696, July 1994.

[32] P. Kotler, L. Brown, S. Adam, and G. Armstrong. *Marketing.* Pearson Prentice Hall, 2004.

[33] J. Lv, K. Li, G. Wei, T. Tang, C. Li, and W. Zhao. Model-based test cases generation for onboard system. In *Autonomous Decentralized Systems (ISADS), 2013 IEEE Eleventh International Symposium on*, pages 1 – 6, Mexico City, Mexico, March 2013. IEEE, IEEE.

[34] N. A. M. Maiden and G. Rugg. Acre: selecting methods for requirements acquisition. *Software Engineering Journal*, 11(3):183–191, May 1996.

[35] L. Martignoni, R. Paleari, G. Roglia, and D. Bruschi. Testing cpu emulators. In *ISSTA '09 Proceedings of the eighteenth international symposium on Software testing and analysis*, pages 261 – 272, New York, NY, USA, 2009. ACM, ACM.

[36] N. Mellegård, M. Staron, and F. Törner. A light-weight defect classification scheme for embedded automotivesoftware and its initial evaluation. In *2012 IEEE 23rd International Symposium on Software Reliability Engineering.* IEEE, 2012.

[37] S. Phillips, J. Sillito, and R. Walker. Branching and merging: An investigation into current version control practices. In *CHASE '11 Proceedings of the 4th International Workshop on Cooperative and Human Aspects of Software Engineering*, pages 9 – 15. ACM, May 2011.

[38] R. Pressman. *Software engineering: a practitioner's approach.* McGraw-Hill Higher Education, 2010.

[39] M. Rochkind. The source code control system. *IEEE Transactions on Software Engineering*, SE-1(4):364 – 370, December 1975.

[40] R. Sasnauskas, O. Landsiedel, M. Alizai, C. Weise, S. Kowalewski, and K. Wehrle. Kleenet: discovering insidious interaction bugs in wireless sensor networks before deployment. In *IPSN '10 Proceedings of the 9th ACM/IEEE International Conference on Information Processing in Sensor Networks*, pages 186 – 196, New York, NY, USA, 2010. ACM, ACM.

[41] R. Wysocki and R. McGary. *Effective Project Management: Traditional Adaptive Extreme*. Wiley Publishing, Inc., 2003.

[42] A. Ying. Why product management is everything. `http://www.inc.com/alan-ying/why-product-management-is-everything.html`, February 2013. [cited 6.5.2013].

[43] V. Zelenty. IEEE recommended practice for software requirements specifications. Standard 830-1998, IEEE Computer Society, 1998.